COMBINATIONAL MACHINE LEARNING CREATIVITY

A Dissertation
Presented to
The Academic Faculty

By

Matthew Guzdial

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Computing

Georgia Institute of Technology

August 2019

COMBINATIONAL MACHINE LEARNING CREATIVITY

Approved by:

Dr. Mark Riedl, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Ashok Goel
School of Interactive Computing
Georgia Institute of Technology

Dr. Charles Isbell
School of Interactive Computing
Georgia Institute of Technology

Dr. Devi Parikh
School of Interactive Computing
Georgia Institute of Technology

Dr. Briak Magerko
School of Literature, Media, and Communication
Georgia Institute of Technology

Dr. Michael Mateas
Computational Media
University of California, Santa Cruz

Date Approved: June 24, 2019

To my parents, my first supporters and first inspirations.

ACKNOWLEDGEMENTS

It is impossible for me to be fully satisfied with these acknowledgements. I have been blessed with incredible support from all possible corners: academic colleagues, industry partners, family, and friends. Let me begin with a blanket statement of thanks for this support, as I fear an exhaustive list would have to be a secondary document.

First and most clearly this work would not have been possible without the support of my advisor. Mark has provided opportunities for my research work, support and feedback when I have needed it, and trust when I have not. In my fellow PhD students I have found community, friends, and academic colleagues that have pushed me to improve. I thank Boyang 'Albert' Li for mentoring me through my first PhD research project and my internship at Disney Research. Alex Zook and Kristin Siu were both incredible guides and collaborators throughout the early years of my PhD, collaborating on both research projects and multiple whiteboards of Pokémon drawings. Brent Harrison and I started in the lab at the same time (though Brent was a post-doc), and I am immensely grateful to have spent three years researching with him. Lara Martin, Chris Purdy, and Upol Ehsan helped reinvigorate the lab and myself, creating valuable research community and becoming valued friends. It's been a joy and an inspiration watching Zhiyu Lin and more recently Raj Ammanabrolu become excellent researchers, and I anticipate even greater future success for the both of them.

I have had the honor of mentoring a rather large number of undergraduates. I greatly appreciate this opportunity and the ways that their work has supported and influenced my own research trajectory. I would especially like to thank Nicholas Liao, who in camping out in the lab one day in an attempt to get involved in undergraduate research indirectly started this aspect of my academic career.

Of course many academics outside of Mark's research lab have impacted my research. In particular, I would like to thank Brian Magerko who took a chance on me as an under-

graduate, and thus launched my interest in research. Thanks as well to Mikhail Jacob, who I first met while working with Brian, and who would go on to be an invaluable colleague. We worked together on the 2018 International Conference on Computational Creativity, and supported each other across research projects, thesis proposals, and thesis defenses.

Outside of Georgia Tech, I would like to thank Gillian Smith, who in an alternate universe was my advisor. Thankfully in this universe I am lucky enough to count Gillian as an excellent colleague, collaborator, and friend. Thanks to Mike Cook for inspiring my research into game generation in the first place and for being an all around great guy. Special thanks to Diana Ford and the entire Unity Research team for believing in my work and granting me a 2018 Unity Graduate Fellowship. I would also like to thank Joseph Osborn, Sam Snodgrass, and Adam Summerville. While we were at separate universities and they've all since graduated, going through parallel PhD programs, collaborating with them on papers, and working with them on the Knowledge Extraction from Games workshop has all been hugely helpful and rewarding.

I could keep going forever. There are so many more people who have impacted my research directly and indirectly. However, my greatest supporters have been my family, who I cannot thank enough. I want to especially thank my parents, Mark Guzdial and Barbara Ericson. They were my very first academic inspirations and their support and advice has been invaluable. But my greatest thanks must be reserved for Jack for their support and patience. They have been my first editor and a springboard for ideas. They have kept me from overworking myself, and if that failed, they took care of me through burn out. I could not have done this without them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Computational creativity is a field focused on the study and development of behaviors in computers an observer would deem creative. Traditionally, it has relied upon rules-based and search-based artificial intelligence. However these types of artificial intelligence rely on human-authored knowledge that can obfuscate whether creative behavior arose due to actions from an AI agent or its developer. In this dissertation I look to instead apply machine learning to a subset of computational creativity problems. This particular area of research is called combinational creativity. Combinational creativity is the type of creativity people employ when they create new knowledge by recombining elements of existing knowledge. All existing computational approaches to represent combinational creativity have relied on human-authored input. Thus I develop a new combinational creativity approach I call conceptual expansion.

The central statement of this thesis is:

For creativity problems that require the combination of aspects of distinct examples, conceptual expansion of generative or evaluative models can create a greater range of artifacts or behaviors, with greater measures of value, surprise, and novelty than standard combinational approaches or approaches that do not explicitly model combination.

By *creativity problems* I indicate problems that require creativity to solve, which I further specify into these problems that require the combination of aspects of distinct examples. I investigate two primary domains that fit the requirements for this type of problem: video game design and image classification, and further investigate it in the domains of video game level design and image generation. Thus by *generative models* I indicate machine-learned models that can generate video game levels, complete video games, or images and by *evaluative models* I indicate models that can evaluate video game level style and classify images. I specifically identify value, surprise, and novelty as they have been

previously identified and applied in prior work as components of creativity, and I indicate measures of these components due to their subjective nature. By standard combinational approaches I mean existing approaches to combinational creativity, and by approaches that do not explicitly model combination I mean appropriate, domain-specific baselines.

This dissertation document covers my novel approach to learn a complete model of a video game design in terms of an existing games level design (how the structure of the game is laid out) and mechanics (the rules that run the game). I then describe my novel combinational creativity approach: conceptual expansion. I demonstrate conceptual expansion on this novel video game model and on learned deep neural network models for image classification and image generation.

CHAPTER 1
INTRODUCTION

Computational creativity is a field that studies issues at the intersection of artificial intelligence and creativity [1]. This includes artificial creativity, the ability for computers to demonstrate behavior that a neutral, human observer would think of as being creative. As a field it stands at odds with more traditional artificial intelligence (AI), particularly machine learning (ML) approaches. While these approaches have shown great success on tasks with large datasets and strong question-answer bindings [2], they often struggle to handle content outside of their training data. I can clarify this statement by defining a class of problems that modern AI and ML approaches would struggle with: *invention problems*.

I characterize invention problems as problems in which the agent cannot achieve a desired solution with its available knowledge. "Available" in this context refers to the information that an agent has access to prior to the problem, not whether this knowledge exists. For example, the ability of an image classifier to correctly label a class it has not been trained on without the meta-knowledge required to identify new classes. Or alternatively the creation of a new class of video game level based on training on existing video game level classes. In both cases these problems are possible to solve, but not without inventing new knowledge, thus: invention problems.

The source of the knowledge does not factor into the invention problem definition, whether encoded by experts or learned from a training set. However, it can be difficult based on the domain to differentiate between intelligence of the system or intelligently encoded human knowledge. For example, in making the choice of what to encode and how to encode it, a human can cut down on the complexity of the invention problem by removing what they think of as extraneous detail. Therefore I prefer machine learned-knowledge for the knowledge base of the invention problem, with the caveat that the learned, initial

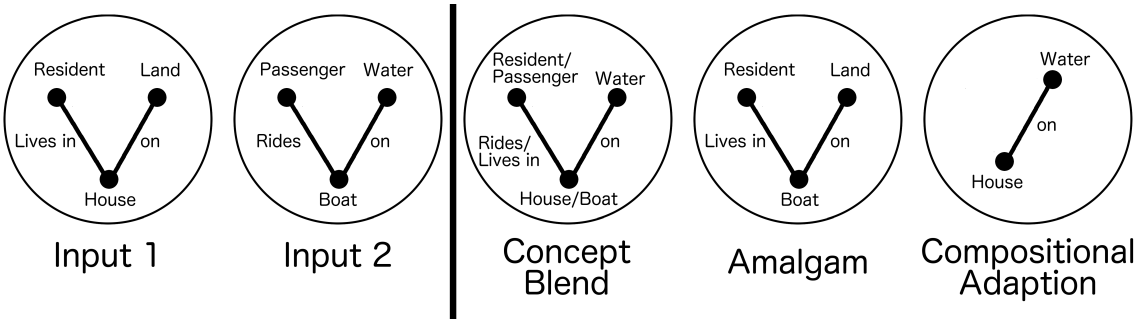

Figure 1.1: Example of three combinational creativity techniques. Two input spaces on left with example output from the three techniques on the right.

knowledge is still insufficient to solve the problem.

This concept of individual invention can be seen as related to Boden's concept of 'p-creativity' [3], a descriptor for output that a creator views as creative, even if the output would not appear creative historically. This class of problem covers a large subset of design, scientific discovery, and general intelligence problems. If it were possible to create general approaches to address invention problems this could allow for the application of artificial intelligence-based solutions to more domains.

Boden defines creativity in terms of surprise, novelty, and value [3]. Given the connections between invention and creativity, these criteria can help to evaluate potential solution techniques to the invention problem in a domain independent way. We can now redefine the invention problem with these criteria for cases without an explicit desired solution: an invention problem is one in which the given knowledge base is insufficient (given non-computational creativity approaches) to achieve a novel, surprising, and valuable answer.

1.1 The Case for Combinational Creativity

The term *combinational creativity*, also called combinatorial creativity, refers to creativity that takes the form of a combinational process merging familiar ideas in an unfamiliar way [4]. The field of computational creativity focuses on research questions formalizing creative problem solving. Within this field there exist many formalized techniques for achieving combinational creativity including conceptual blending [5], amalgamation [6],

and compositional adaptation [7]. At a high level these techniques allow the recombination of items from within a knowledge base to create new items, while retaining some of the structure from the parent items.

Figure 1.1 gives examples of illustrative output from three combinational creativity techniques given the two input concepts, represented as graphs, on the left. The inputs represent encodings for the concept of a house (Input 1) and a boat (Input 2) in terms of their uses and locations. The three combinational creativity techniques are: *conceptual blending*, *amalgamation*, and *compositional adaption*. All three can be generally understood as functions that take some number of input concepts and a knowledge base, and return novel concepts. Conceptual blending merges elements that are similar to one another according to the knowledge base, but can include non-merged elements as well, thus leading to the "Houseboat" blend [8]. Amalgamation incorporates as much information as possible from both input spaces without merging, instead choosing between sufficiently similar elements. Compositional adaption breaks apart the individual elements of each input and composes novel concepts piece-by-piece, which allows it to generate smaller, novel concepts [9]. The output concepts illustrated in Figure 1.1 are a single output of many possible combinations from each technique.

Notably there are differences in the types of new knowledge created by each of the three approaches. Amalgamation cannot create new components, edge or node values in the example of Figure 1.1, but can connect previously unconnected components, forming novel concepts. Compositional adaptation similarly cannot create new components, but can create concepts with novel global structure (e.g. new graph structure). Conceptual blending is the only approach of the three that can create new components, but cannot vary structure like compositional adaptation.

Combinational creativity techniques are well-suited for invention problems. The connections between invention and creativity are self-evident, but beyond that this set of techniques are modeled on human problem solving processes [5]. Further, given that the tech-

niques can generate a space of combinations they have an advantage over techniques that generate a single optimal answer for the criteria of creativity over time.

1.2 The Case for Combinational Machine Learning Creativity

Combinational creativity techniques cannot create new knowledge out of thin air; they require a knowledge base of concepts or cases. This knowledge base has historically been encoded by domain experts. Due to this authorial input, it is difficult to demonstrate that the success of an agent arose from the combinational creativity technique and not creative knowledge encoding from the domain expert. For example, consider the convenient similarity in structure of the inputs in Figure 1.1. As an alternative one could consider automatically deriving a knowledge base through machine learning techniques. One reason machine learning is not applied to combinational creativity techniques is that such techniques historically have required well-formed knowledge as input. Most combinational creativity techniques could not easily adapt to the messy input of most machine learning techniques. Beyond the knowledge acquisition issue, that combinational creativity techniques can produce a space of possible outputs rather than a single answer has historically been considered a drawback. Computational creativity researchers have developed heuristics, rules, and processes to limit the space of acceptable outputs [10]. This assumption, by its nature, limits the output and potential applications of such techniques.

1.3 The Case for Video Games as Domain

Much of this dissertation focuses on the domain of video games. A particular subset of video games can be understood as a simplified version of reality, those with a consistent structure and underlying physics system. The nature of this class of games allows one to anticipate that techniques developed in this domain could apply to more complex domains. This notion has been the primary motivation behind automated game playing research, varying from retro Atari games with Deep Mind [11], to board games such as Go [12], and

more modern games like DOTA2 [13] and Starcraft 2 [14].

Video games of this type can be understood as being composed of two things: (1) a series of challenges across a single space referred to as a level, and (2) the rules that determine how the game elements interact. Solving the invention problem in the domain of video game design requires a knowledge base composed of models of both these categories of knowledge.

1.4 Thesis Statement

I focus on approaches to solve invention problems in two primary domains: video game design and image classification. In particular I address these problems through the development of a novel combinational creativity approach I call *conceptual expansion*. This approach is designed from the start to work with messy, machine learned-knowledge as its input and to address several other key limitations of historic combinational creativity approaches.

Thus in this dissertation I investigate the following statement:

> For creativity problems that require the combination of aspects of distinct examples, conceptual expansion of generative or evaluative models can create a greater range of artifacts or behaviors, with greater measures of value, surprise, and novelty than standard combinational approaches or approaches that do not explicitly model combination.

By "creativity problems" in the above I mean problems that an outside observer would consider to require creativity. I then focus on the sub-domain of problems require the combination of aspects of distinct examples, which can be considered analogous to the invention problems defined above.

The "generative and evaluative models" in the above refers to machine learned models

that either generate or evaluate content. For example models that can generate images or video game levels (generative models) or those that can classify images or evaluation video game levels (evaluative models).

Value, surprise, and novelty have been identified by Boden as important elements of creativity. I specifically identify "measures of" these elements as they are not possible to objectively measure in some domains.

For this dissertation I define value as a domain dependent measures of utility. In the video game domain this might include subjective measures like appropriate challenge, engagement, and fun. For an image classifier one can instead draw on the objective measure of accuracy. I define surprise as the amount to which the behavior of a model defies user expectation. Novelty I define as the degree to which the behavior of a model differs from a training, inspiration set, or initial knowledge base. Therefore, for example in the case of a machine learning system for generating computer games a low-surprise, high-novelty game would be a game that was an average of its training set. Comparatively a high-surprise, low-novelty game might be a game composed of a novel combination of uncommon elements from existing games.

I identify conceptual expansion, amalgamation, and compositional adaptation as "standard combinational approaches" due to the fact that they are all domain-independent combinational creativity approaches with distinct outputs from one another that have a body of prior work investigating their applications. I compare my approach against these for problems in which the input can be treated as composed solely of symbolic features. By "approaches that do not explicitly model combination" I indicate domain-specific baselines, which I compare against as appropriate.

1.5 Outline

In Chapter 2 I discuss prior work related to this dissertation. In order to address the video game invention problem one would first need learned models of video game level design

and video game rules, as these two elements are the minimum needed to represent a game. I discuss learning models of level design in Chapter 3 and models of a game's mechanics in Chapter 4.

Learning models of game design has been covered previously in the field of Procedural Content Generation via Machine Learning (PCGML) [15]. Moving beyond this area of work, I drew on conceptual blending to combine the machine learned models of level design to create models capable of generating novel, surprising levels. I describe this work in Chapter 5. This stands as a case study in the application of combinational creativity to machine-learned models. Leveraging conceptual blends leads to generated models capable of better evaluating novel and surprising levels from the original human level designer, representing a limited solution to the invention problem within the domain of level design.

In Chapter 6 I introduce my novel combinational creativity technique: conceptual expansion. I demonstrate conceptual expansion's superiority in the range of its output space through a comparison with three existing combinational creativity techniques for the video game level invention problem. In Chapter 7, I extend this to generate entire video games, including both objective and human subject study evaluations. I demonstrate the application of conceptual expansion in non-game domains in Chapter 8, recombining image classification and generation deep neural networks (DNNs), Unlike for video games I do not need to develop new approaches to learn deep neural networks. Conceptual expansion applied to DNNs out-performs standard transfer learning and few-shot learning baselines.

I end this document with Chapter 9, stating conclusions, contributions across the areas of PCGML and computational creativity, and potential areas of future work.

CHAPTER 2
SITUATING THE WORK

In this chapter I summarize prior related work in the development of artificial intelligent game designers, machine learning focused on games and creativity, and computational creativity. Notably I focus on the most relevant prior work, and especially prior work that relates to my own in technique or subject matter.

2.1 Computational Creativity

The field of computational creativity represents the overlapping of fields like artificial intelligence and cognitive science, concerned with problems that one would consider requiring creativity if attempted by a human [1]. Combinational creativity represents a sub-field of computational creativity.

2.1.1 Boden's Theory of Creativity

Boden defines creativity in terms of surprise, novelty, and value [3]. Given the connections between invention and creativity, these criteria can help to evaluate potential solution techniques to the invention problem in a domain independent way.

Boden defines three particular forms of creativity [16]: *combinational*, *exploratory*, and *transformative*. In combinational creativity two existing exemplars have their elements or features combined. In exploratory creativity an existing conceptual space is explored for novel exemplars (e.g. a new chair in a design space of all possible chairs). In transformational creativity a conceptual space is transformed (e.g. redefining what it means to be a chair). The definition of combinational creativity used in this paper does not cleanly fit into any one of these categories. This is due to the fact that I focus on systems with learned conceptual spaces that then recombine elements not of exemplars from these spaces but

recombine these conceptual spaces themselves. It may be argued that this represents transformational creativity. However I focus explicitly on defining new conceptual spaces via combination and not through any other strategy. Thus I instead extend the definition of combinational creativity given by Boden to include recombining approximations of conceptual spaces (machine learned models and representations). In Chapters 7 and 8 I employ a search-based approach to explore the space of possible combinations, thus it could also be argued that these particular approaches represent exploratory creativity.

Boden identifies two distinct evaluative perspectives one can take when it comes to creativity [17]: *p-creativity* and *h-creativity*. P-creativity stands for personal creativity. In other words something that would be viewed as creative (e.g. novel, valuable, and surprising) from a particular individual, whether or not it had previous existed elsewhere (perhaps not truly novel). Comparatively h-creativity is truly novel creativity, some artifact, action, or idea that has never existed before. H-creativity is, of course, incredibly difficult to identify and can often be misattributed based on the background of the particular person doing the evaluation. Therefore for this work I focus entirely on p-creativity. In other words, I judge the output of a system as creative in terms of the knowledge base it had access to (e.g. is it novel, surprising, and valuable compared to its knowledge base?).

2.1.2 Discovery Systems

Discovery Systems refers to attempts to create new knowledge artificially [18]. In the context of this document, these systems can be thought of as being built to solve specific invention problems. The most well-known and successful discovery system is Lenat's AM [19], which output helpful mathematical concepts. However, despite initial praise, AM did not lead to an era of "super-intelligent artificial mathematicians" [18]. Lenat would go on to build Eurisko [20] in attempt to explore AM's shortcomings. AM and Eurisko are related to my work as they also took in some input and tried to create novel output, though primarily through exploratory creativity (heuristic search) instead of combinational creativity. Lenat

eventually concluded that AM's success had more to do with how knowledge was encoded, which meant that even his random operators could lead to useful output [21]. Essentially Lenat, via his authoring of the operators, initial input data, and chosen representation, had authored a space of possible output that was sufficiently rich that it was trivial to find high quality output. I demonstrate results that relate to this in a random baseline outperforming other combinational creativity approaches with high quality, hand-authored input in Chapter 6.

2.1.3 Analogical Reasoning

Analogical reasoning is the study of the ability in humans to draw analogies [22], structural metaphors that connect two distinct domains. In terms of computational creativity, computational models of analogical reasoning have been applied in similar problem domains as combinational creativity in terms of inventing new knowledge via reuse of old knowledge. While there are many implementations [23], perhaps the best known computational analogical reasoning approach is Falkenhainer et al.'s Structure-Mapping Engine [24]. In this approach a "base" domain is generalized and then adapted into a "target" domain with the purpose of creating innovative new knowledge. Analogical reasoning has shown success in a number of domains, but is typically limited to modifying a single hand-authored representation and requires a domain-specific function or knowledge representation for generalization. Thus, given my focus on combinational creativity I do not engage directly with this concept in this work. However, transfer learning can be understood as similar to analogical reasoning, which I address in Chapter 8.

Analogy-based design focuses on the application of analogiocal reasoning in design problems [25], which notably includes the invention of new designs given a corpus of old designs. For example a recent example of analogy-based design is the Design Evaluation through Simulation and Comparison (DESC) AI agent [26]. This agent looks to aid in the creation of novel designs in the context of biologically-inspired design [27], a type

of analogy-based design. The design models are placed in a Structure-Behavior-Function [28] representation, which can be understood as similar to the game graph representation I present in Chapter 7. The structure relates to the construction of a device, the behavior relates to what a device can do, and function relates to how a device can do it. All of the game graphs I make use of have the same behavior in this framework, but differ in terms of their level design knowledge (structure) and rules (function).

2.1.4 Combinational Creativity

One common feature across combinational creativity techniques is the need for some ability to map between elements of individual cases [24, 29]. *Analogical reasoning* has traditionally been one of the leading conceptual mapping approaches, as it maps concepts based on relative structure instead of surface features [30]. This type of structural mapping has proven popular as it tends to better match human problem solving [27, 31]. However, such analogical reasoning systems require a non-trivial amount of human input, as a human author must encode concepts in terms of their structure and how to compare structural information within a domain. As an alternative I focus on work that automatically learns the structure for a given domain.

There have been many approaches to combinational creativity over the years, which I will attempt to briefly summarize. *Case-based reasoning* (CBR) represents a general AI problem solving approach that relies on the storage, retrieval, and adaptation of existing solutions [32]. The adaptation function represents a large class of combinational creativity approaches, which one can place into the categories of substitutional adaptation and structural adaptation [7, 33]. Essentially, whether components are substituted or whether the structure of the case is adapted. These techniques tend to be domain-dependent. For example for particular domains like text generation or tool creation, focusing on rules, heuristics, or constraints that encode the domain-specific knowledge needed for adaptation [34, 35]. Murdock and Goel [36] combine reinforcement learning with case-based reasoning, which

aligns with my work combining computational creativity and machine learning research.

The area of belief revision, modeling how beliefs change, includes the function to merge prior existing beliefs with new beliefs for a variety of domains [37, 38, 39, 40, 41]. Belief merging has been applied to CBR as an adaptation function [42], further aligning it as a combinational creativity technique. However, due to the focus of belief merging the process ends with single final merged belief state, while I focus on spaces of combinations. The mathematical notion of convolution has been applied to blend weights between two neural nets in work that parallels my desire to combine computational creativity and ML, but without conclusive results [43].

Combinational creativity algorithms tend to have many possible valid outputs. This is typically viewed as undesirable, with general heuristics or constraints designed to pick a single correct combination from this set [44, 6]. This limits the potential output of these approaches, I instead employ a domain-specific heuristic criteria to find an optimal combination from a space of possible combinations.

Genetic algorithms, also referred to as evolutionary methods, have traditionally proved popular with computational creativity work [45]. Part of this can be understood as due to the inclusion of the crossover operator [46], which takes typically two inputs search points and combines them to produce new search points. The crossover operator can therefore be understood as a combinational creativity function. However, these operators are nearly always domain-dependent. At best they may be general to a particular representation, such as a one-dimensional array [47].

I identify three specific existing combinational creativity techniques for deeper investigation, due to the fact that they are well-formed in domain-independent terms, and output distinct and large spaces of combinations: conceptual blending, amalgams, and compositional adaptation.

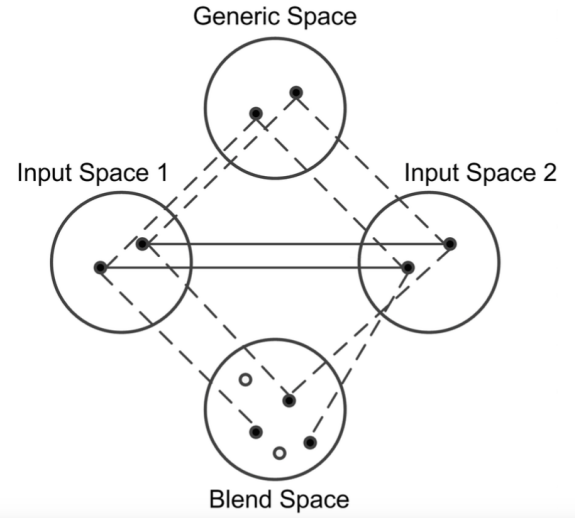

Figure 2.1: Representation of the canonical four space conceptual blending approach from [48].

2.1.5 Conceptual Blending

Fauconnier and Turner [48] formalized the "four space" theory of conceptual blending. In this theory they describe four spaces that make up a blend as seen in Figure 2.1: two *input spaces* represent the unblended elements, input space points (or features) are projected into a common *generic space* to identify equivalence, and these equivalent points are projected into a *blend space*. In the blend space, novel structure and patterns arise from the projection of equivalent points. Fauconnier and Turner [48, 5] argued this was a ubiquitous process, occurring in discourse, problem solving, and general meaning making.

Conceptual blending typically requires a large amount of human authoring for individual conceptual spaces. O'Donoghue et al. [49] looked into deriving this knowledge automatically from text corpora, producing graphical representations of nodes and their verb connections. My work in Chapter 5 runs parallel to O'Donoghue et al., but in the domain of video game levels. There has also been work in blending individual tagged exemplars together based on surface level features of components [50].

Fauconnier and Turner originally developed a set of heuristics for domain-independent measures of quality for blends. As an alternative more recent work has introduced goals for blends [51]. I make use of goals in my work applying conceptual blending to level design

and video game design.

2.1.6 Amalgamation

Ontañón designed amalgams as a formal unification function between multiple cases [6]. Similar to conceptual blending, amalgamation requires a knowledge base that specifies when two components of a case share a general form, for example *French* and *German* can both share the more general form *nationality*. Unlike conceptual blending, this shared generalization does not lead to merging of components, but requires that only one of the two be present in a final amalgam. For example, a *red French car* and an *old German car* could lead to an *old red French car* or an *old red German car*.

Amalgams have been utilized as the adaptation function in CBR systems [52], combined with conceptual blending for product development [53], and adapted to an asymmetrical form for story generation [54]. Amalgamation represents a strong general method for combinational creativity, though it suffers from the same drawbacks of other adaptation methods in terms of a traditional reliance on authored knowledge bases and a domain-specific generalization function.

2.1.7 Compositional Adaptation

Compositional adaptation arose as a CBR adaptation approach [55, 33], but has found significant applications in adaptive software [56, 57]. The intuition behind compositional adaptation is that individual component cases can be broken apart and reused based on their connections. In adaptive software this takes the shape of sets of functions with given inputs and outputs that can be strung together to achieve various effects, which makes compositional adaptation similar to planning when it includes a goal state or output. However, it can also be applied in a goal-less way to generate sequences or graphs of components.

Compositional adaptation has been applied to recipe generation [9, 58], intelligent tutoring systems [59], and to traditional CBR approaches [60]. Unlike amalgamation and

conceptual blending, compositional adaptation does not require an explicit knowledge base by default. However, it is common to make use of a knowledge base to generalize components and their relationships in order to expand the set of possible combinations. As with the prior combinational creativity approaches, this knowledge base is almost always human-authored.

2.2 Procedural Content Generation

Procedural content generation is the umbrella term for systems that take in some design knowledge and output new assets from this knowledge. Approaches include evolutionary search, rule-based systems and instantiating content from probability tables [61, 62]. The often cited goal of these systems is to reduce the authorial burden on designers. However, these systems tend to only transfer the burden from creating content to supplying the design knowledge needed to create that content.

2.2.1 Computational Creativity and Games

Combinational creativity and creativity research in general is rarely applied to video games [63]. Dormans and Leijnen [64] proposed a number of applications of computational creativity theory to games. Some prior work has looked into knowledge intensive conceptual blending systems to create new elements of video games such as sound effects and 3D models [65, 66]. The Game-O-Matic system made use of concept mapping to match verbs onto game mechanics to create arcade-style games based on human-authored mapping knowledge [67]. Gow and Corneli [68] proposed a system to generate small games via conceptual blending. Permar and Magerko [69] presented a system to produce novel interactive narrative scripts via conceptual blending, using analogical reasoning.

Snodgrass and Ontañón [70] leveraged transfer learning to train a machine learning approach on levels from two separate platform games, meant to address the problem of a lack of training data, but with the effect of combining level structure in the generated levels.

My work looks to explicitly combine data from multiple games. Sarkar and Cooper [71] combine the output of two deep neural networks trained to generate levels for two different games in a process inspired by conceptual blending. I focus on the combination of learned models, not of the output of learned models.

2.2.2 Procedural Content Generation via Machine Learning

Much of my work falls within the sub-field of procedural content generation via machine learning (PCGML) [15], which encompasses a range of techniques for generating content from models trained on prior content. Super Mario Bros. has been the most consistent domain for this research, with researchers applying such techniques as markov chains [72], Monte-Carlo tree search [73], long short-term recurrent neural networks [74], autoencoders [75], generative adversarial neural networks [76], and genetic algorithms through learned evaluation functions [77]. Closest to my own work in terms of combining computational creativity and machine learning, Hoover et al. [78] adapt computational creativity techniques originally developed to generate music to create Super Mario Bros. levels.

The major differences between my research and the other work in the field of PCGML are (1) the source of training data and (2) the focus on creativity. In terms of the first, by focusing on gameplay video as the source of training data I avoid the dependence on existent corpora of game content, opening up a wider set of potential games on which to apply my techniques. Further, these approaches require substantial human authoring, such as authored patterns of level content or categorization of stylistically similar level elements (e.g. many different enemies abstracted as a single "enemy" class) [79]. This raises the question as to whether the creativity in and success of PCGML systems arises from the system or the system designer's intuition.

Second, my work focuses on creative generation of game content, while most prior PCGML techniques rely on learning a statistical distribution and generating new content from within this distribution. For example, none of the prior techniques would be able to

generate a novel level type not seen in an existing game such as an underwater castle level for the game Super Mario Bros., as I demonstrate in Chapter 5.

2.2.3 Game Rules

To my knowledge, my work is the first that includes the generation of video game rules from machine learned models. However, there is a significant body of prior work generating game rules through more traditional procedural content generation techniques, running an optimization process on a human-authored game engine or rule space [80]. Togelius [81] presents an iterative development paradigm for a human player to create a game engine during play, as an intelligent system attempts to learn the causes behind user-specified effects to create rulesets. Both Cook et al. [82], and Zook and Riedl [83] make use of approaches that generate player character game rules within human-authored game engines, and verify these rules with forward simulation.

Game rule generation has historically existed in the absence of generated structure, creating new rulesets for existing level designs. The majority of prior approaches to game rule generation have relied upon authoring a general game rule representation that is then constructed via grammars [84], optimized [85, 86, 87, 82] or constrained [88, 83].

The General Video Game Rule Generation track [89] serves as a competition for ruleset generators in which generators are given a level and must create an appropriate ruleset. Khalifa et al. [89] introduced two initial generators, a constructive approach based on human-authored rules and a genetic algorithm approach.

2.2.4 Automated Game Design

Treanor et al. [67] introduced Game-o-matic, a system for automatically designing games to match certain arguments or micro-rhetorics [90]. This process created complete, if small, video games based on an authored procedure for transforming these arguments into games. Cook et al. produced the ANGELINA system for automated game design and development

[91]. There have been a variety of ANGELINA system versions, each typically focusing on a particular game genre, with each employing grammars and genetic algorithms to create game structure and/or rules [82].

Nelson and Mateas [92] introduce a system to swap rules in and out of game representations to create new experiences. This can be understood as domain-specific example of combinational creativity. Nielsen et al. [93] introduced an approach to mutate existing games expressed in the video game description language (VGDL) [94]. Nelson et al. [95] defined a novel parameterized space of games and randomly alter subsets of parameters to explore this design space.

More recent work has applied variational autoencoders to produce noisy replications of existing games, called World Models or Neural Renderings [96, 97]. By its nature this work does not attempt to create new games, but recreate subsections of existing games for automated game playing agent training purposes. In addition, it is limited thus far to these small subsections of existing games.

The approaches listed in this section rely on creative combinations or parameterization of designer-authored or crowdsourced [98] representations of game knowledge. In my work I focus on machine-learned game representations. Osborn et al. [99] propose automated game design learning, an approach that makes use of emulated games to learn a representation of the game's structure [100] and rules [101]. This approach is most similar to my work in terms of deriving a complete model of game structure and rules. However, this approach depends upon access to a game emulator and has no existing process for creating novel games.

2.3 Machine Learning

In this section I briefly cover a set of machine learning subfields with varying degrees of relevance to my own work. Object and scene modeling comes from the graphics field, and inspires the technique I used to model game level design knowledge. Machine vision

represents the problem of deriving knowledge from images, which I apply in a limited capacity to pull information from gameplay video. My work on learning game engines overlaps with work in automated game playing, automated understanding, and automated game understanding.

2.3.1 Object and Scene Modeling

The problem of generating game levels from gameplay video is related to object modeling from exemplars, in which a set of exemplar objects are used to train a generative model [102, 103, 104]. My level design learning approach builds off these techniques, which also use probabilistic graphical models. However, a majority of object modeling approaches require human-tagging of individual elements and how they fit together. My model does not require any human tagging, instead relying on machine vision and probability to smooth any incorrect tags.

2.3.2 Machine Vision

Machine vision is not often applied to computer games. However, approaches exist to train agents to play Atari games from pixel input [11, 105]. While these approaches and my own use machine vision to process pixels over time, my system focuses on extracting design principles and rules instead of learning to play games. Since my original work applying machine vision to computer games for procedural content generation discussed in Chapter 2, there has been a large body of follow-up work looking at extracting player paths [106, 107] and deriving game maps [108, 100].

2.3.3 Automated Game Playing

Automated game playing has stood as a goal for artificial intelligence from its earliest days [109], and has had great success in classic games like chess and go. Most closely related to this work are approaches to utilize pixel input to learn to play retro video games.

For example, Mnih et al. [110] used deep convolutional networks to learn how to the play Atari games from pixel input. Later work has brought automated game playing to Doom [111], and more modern games such as Minecraft [112]. The end models of these approaches and my work on learning rules from video differ. For example, Mnih et al.'s extracted model will recognize the action for the player to activate in each state, and will ignore any elements that do not impact reward (score) and those not currently visible. On the other hand, my rule learning system includes decorative elements and can reason about the off-screen effects of actions (for example, killing an enemy offscreen by recognizing when an enemy does not appear when expected).

2.3.4 Automated Understanding

Automated understanding, also known as common sense learning [113] or hypothesis generation [114], is the problem of taking a sequence of events, building some model that explains the events, and using the model to predict future events. A common modern approach to this problem relies on convolutional neural nets, learning from sequences of images [115, 116]. These techniques take in a frame and predict the next frame in a sequence, without modeling the underlying effects. Perhaps closest to my work, Selvaraju et al. [117] derive an explanation for a convolutional neural net's prediction, without learning an explicit model of the physics at play in the images whereas the rules my system learns from video implicitly include explanations.

An alternate approach to automated understanding common to reinforcement learning techniques is *forward model learning* [118]. In forward model learning a transition function is learned that allows an agent to make predictions in a simplified state space from sequences of state changes. This approach has been applied to navigation control [119], video games such as Starcraft [120] and arcade-like games [121]. Closest to my work in rule learning from video, Ersen and Sariel [122] derive a model of the environment for a puzzle game composed of formal logic rules based on hand-authored sequences of events.

My work differs from typical forward model techniques as it learns from and makes predictions in a pixel-level state space.

2.3.5 Automated Game Understanding

The field of automated game understanding is much more recent than automated game playing. Martens et al. [123] made use of proceduralist readings to derive possible meanings from specially constructed representations of games. Sumerville et al. [124] demonstrated the ability to automatically derive object characteristics (e.g. "hurts player", "is killed by player") from gameplay logs. More recent work has focused on learning a model of player movement derived from gameplay video [107] or emulators [125, 101].

My work differs from these prior techniques by learning an entire game engine, capable of forward simulation and by deriving these models from video rather than gameplay logs. Despite this work being the first to attempt to learn game engines in this manner both Gow and Corneli [68] and Summerville et al. [15] propose this problem and suggest possible solutions to it.

2.3.6 Knowledge Reuse in Neural Networks

A wide range of prior approaches exist for the reuse or transfer of knowledge in neural networks, such as zero-shot, one-shot, and few-shot learning [126, 127], domain adaptation [128], and transfer learning [129, 130]. These approaches either require an additional set of features for transfer, or depend upon backpropagation to refine learned features from some source domain to a target domain. In the former case these additional transfer features can be hand-authored [129, 131, 132] or learned [133, 134, 135, 136, 137]. In the case of requiring additional training these approaches can freeze all weights of a network aside from a final classification layer or can tune all the weights of the network with standard training approaches [138, 139]. As an alternative one can author an explicit model of transfer such as metaphors [133] or hypotheses [140]. To the best of my knowledge I was the

first to attempt few-shot training of generative adversarial networks (GANs). There have been more recent approaches to few-shot training of generative adversarial neural networks, but they focus on alterations to model architecture instead of knowledge reuse [141, 142]. Further, some work exists at exploring the space between distributions of classes [143].

Kuzborskij et al. [144] investigate the same n to n+1 multiclass transfer learning problem as my image classification experiments, and make use of a combination of existing trained classifiers. However, their approach makes use of Support Vector Machines with a small feature-set and only allows for linear combinations. Rebuffi et al. [145] extended this work to convolutional neural nets, but still requires retraining via backpropagation. Chao et al. [146] demonstrated that average visual features can be used for zero-shot learning, which represents a domain independent zero-shot learning measure that does not require human authoring or additional training.

One alternative to reusing learned knowledge in neural networks is to extend a dataset to new classes using query expansions and novel data downloaded from the web [147, 148]. However, I am interested in problems in which no additional training data exists. Neuroevolution is an approach to train neural networks via evolutionary search, which includes an explicit recombination step [149]. However, this approach does not transfer knowledge from one domain to another.

CHAPTER 3
AUTOMATED LEVEL DESIGN

Procedural level generation is the problem of generating high-quality game levels automatically. To address the video game invention problem, we will need to create levels for the invented games. Existing level generation systems typically employ rule-based methods or formulate optimization problems, where an expert designer encodes domain-specific knowledge as a set of rules, constraints, and/or objective functions. The process of encoding this knowledge is time-consuming and, by its nature, includes the biases of the encoder. This is desirable when the encoder is the game designer, but this is not always the case. As an alternative, data-driven approaches learn a rule set or optimization function from exemplars. However, when it comes to game levels, static exemplars like level maps are insufficient as they lack a record of *how* the level is played.

I propose an alternative to these approaches by learning a generative model from representations of player *experience*: gameplay videos. Given a representation of player experience, a system can learn a generative model of an interactive scene without any additional authored knowledge, as the representation includes how a player interacts with the scene. Learning a level design model allows a system to generate novel levels more in the style of the original levels as the model does not include encoder biases. The system encodes "bias", to an extent, but it is the original expert designer's biases or design style that is encoded rather than those of the algorithm's authors. I therefore advocate for this approach in empowering novice game designers to make game levels for use in education, training, and entertainment, as expert design knowledge is required to either guide novices or fully automate design. In addition, these same features make this process ideal for deriving level design knowledge for a fully machine learned representation of a video game, which I require for the video game invention problem.

I present a system that learns a generative, probabilistic model from gameplay video exemplars in an unsupervised fashion. At a high level the system parses raw gameplay video as input, categorizes level sections based on their contents and the player's behavior, and learns a probabilistic graphical model of spatial relationships of content. This model captures the probabilistic structure of level components, which can then be used in the generation of novel content that stylistically matches the original exemplars. I evaluate this approach in a human subject study comparing the quality of output models to other machine learned models of level design and determining the extent to which the model's learned evaluation function correlates with human evaluations.

3.1 Background

In this section I situate the work presented in this chapter in terms of prior work. As stated above, *procedural content generation* is the umbrella term for systems that take in some design knowledge and output new assets from this knowledge. However this work focuses on the subfield of procedural content generation via machine learning (PCGML) [15]. As in this chapter, Super Mario Bros. has been the most consistent domain for this research, with researchers applying such techniques as markov chains [72], monte-carlo tree search [73], long short-term recurrent neural networks [74], autoencoders [75], generative adversarial neural networks [76], and genetic algorithms with machine-learned evaluation functions [77]. The primary differences between the work presented in this chapter and these related work are: (1) the source of training data and (2) the basis of the approach. In this chapter I employ gameplay video as the input training data while these related works instead employ hand-authored representations of Super Mario Bros. levels. The system in this chapter also employs a component-based probabilistic graph, which has not appeared across any prior attempt to generate Super Mario levels. Further, these prior approaches require substantial human authoring, such as authored patterns of level content or categorization of stylistically similar level elements (e.g. many different enemies abstracted as "enemy") [79].

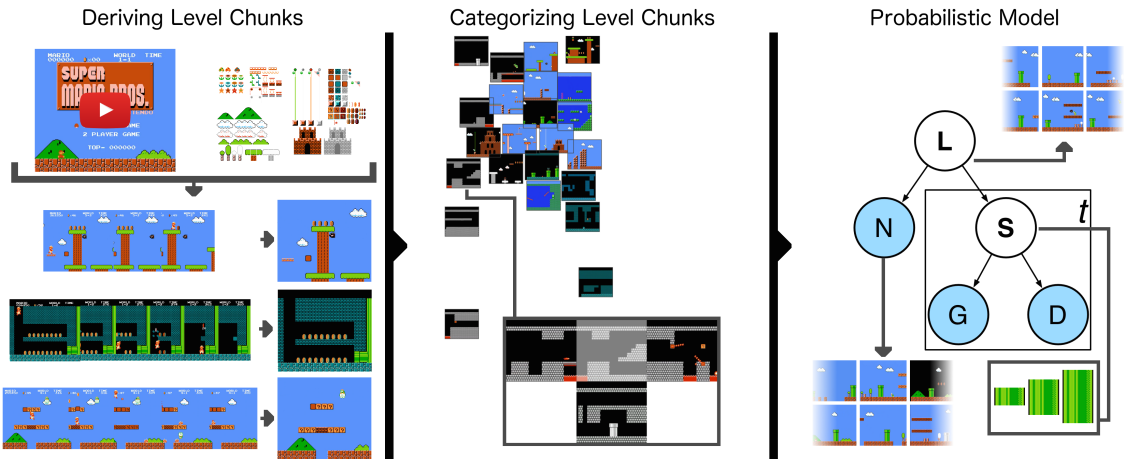

Figure 3.1: A visualization of the full level design learning system.

I make use of two of these prior approaches in the evaluation as they were the only approaches with public, published results at the time of the user study. Dahlskog and Togelius [77] extracted vertical slices of levels from Super Mario Bros. levels to inform a heuristic function in an evolutionary process. Snodgrass and Ontañón [150] trained a hierarchical Markov chain on Super Mario Bros. levels, on both tile-by-tile and abstract tile transitions.

The problem of generating game levels from gameplay video is related to object modeling from exemplars, in which a set of exemplar objects are used to train a generative model [102, 103, 104]. My approach builds off these techniques, which also use probabilistic graphical models. However, a majority of object modeling approaches require human-tagging of individual object features and how they fit together (e.g. arms, legs, etc.). My model does not require any human tagging, instead relying on machine vision and probability to smooth any incorrect tags.

3.2 System Overview

The system can be understood as containing three parts, operating sequentially. I include a visualization of the full system in Figure 3.1. First the system automatically derives sections of level from video. Second the system automatically categorizes these sections.

Third, the system derives probabilistic graphical models from each category in a process inspired by Kalogerakis et. al [102]. After this process, the system can generate novel level chunks and combines these chunks based on learned transitions to create full levels.

I begin by supplying the system with two things: a set of videos and a spritesheet. This input is simple to gather with the advent of "Let's Plays" and "Long Plays" [151]. By spritesheet I indicate the set of "sprites" or individual images used to build up the levels of a 2D game. For this work I utilized nine gameplay videos and a fan-authored spritesheet.

3.3 Model Learning

The system learns a generative, probabilistic model of shape-to-shape relationships from gameplay videos. These kinds of models, common in the object modeling field, require a set of similar objects as input. Given that the input to the system is gameplay video, I needed to determine two things: (1) what input the probabilistic model should learn from and (2) how to categorize this input in an unsupervised fashion to ensure the required similarity. I chose to make use of *level chunks*, sections of level composed of geometric sprite data and learned tags of player information, as the basis of the model. I categorize these level chunks with K-means clustering, and each learned category is then used as input to learn a generative, probabilistic model.

3.3.1 Defining and Categorizing Level Chunks

I employ OpenCV [152], an open machine vision library, to parse the input set of gameplay videos frame-by-frame with the associated spritesheet. The output of this parsing is a list of each individual sprite and their positions per frame. From this list, individual frames join together into level chunks when adjacent and if they share 90% of the same content. If a frame differs entirely from the next, its level chunk is marked as the end of a level. This is required for the sequencing of generated level chunks into full levels that I describe below. Along with a list of sprites and their positions each level chunk stores an *interaction*

time value equivalent to the number of frames that combined to form the level chunk. This allows the system to capture how long a player remained in each level chunk. Interaction time allows the system to do without manually encoded design knowledge representing difficulty or reward. Instead the interplay between the sprites and interaction time in a chunk allows the system to automatically separate these experiential elements. For example, a level chunk with a large amount of coins and a high interaction time is likely rewarding. With nine gameplay videos as input this process found 13,492 level chunks.

The system utilizes K-means clustering to categorize the learned level chunks, with K estimated via the distortion ratio [153]. I designed a two-tier clustering approach. For the first round, each level chunk is represented as an n-dimension vector of counts per sprite type (e.g. ground, block, coin) and normalized interaction time. I chose to normalize interaction times for each gameplay video in order to minimize the impact of player skill differences. The system employs Euclidean distance for the K-means distance metric. For the nine gameplay videos this process found twenty-three initial categories.

The first round of clustering allows the system to derive a measure of sprite relevance in the second round of clustering. This makes up for the fact that the system does not have design knowledge to determine noteworthy sprites, such as power-up blocks or unusual enemies. I base this measure on a variation of term frequency-inverse document frequency or TF-IDF. TF-IDF is typically utilized in natural language processing and search engines to determine the relevance of a term in a document. Formally:

$$relevance(t, d, D) = f_{t,d} * log(N/n_t) \tag{3.1}$$

Where t represents a particular sprite type, d represents a particular level chunk, D represents an entire category of level chunks, N represents the number of level chunks in the category and n_t represents the number of level chunks where the sprite type t appears. $f_{t,d}$ represents the raw number of times a sprite of type t occurs in level chunk d. In this way a

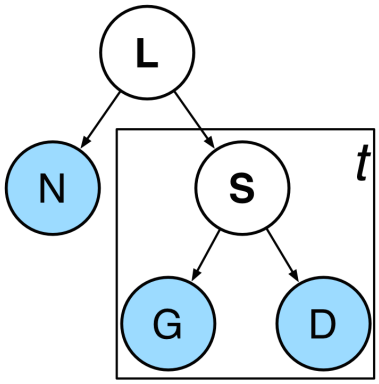

Figure 3.2: A visualization of the abstract probabilistic graphical model.

notion of relevance can be determined per sprite type per category, which would not be as useful for the system if it used this metric prior to having initial clusters. Along with these relevance scores each level chunk is represented according to its normalized interaction time and a value representing its normalized position in its level. Using the Euclidean distance metric once more for this n-dimensional vector I reclustered each of the categories, finding sixty-nine second round clusters, with each of the initial first round clusters splitting into an average of three second round clusters.

3.3.2 Probabilistic Graphical Model

The system builds a probabilistic graphical model from each of the level chunk clusters, which represents styles of relative sprite placements. The intuition for this per-category learning is that different types of level chunks will have different relationships, and that therefore different models must be learned on an individual category basis. The model extracts values for latent variables to represent probabilistic design rules. Figure 3.2 shows the probabilistic model using the standard "plate" visualization. White nodes represent hidden variables, with the blue nodes representing the observed variables taken directly from the level chunks in a category.

The three observable nodes are the G node, D node, and N node. The G node in the graphical model represents the sprite "geometry", an individual shape of sprite type t.

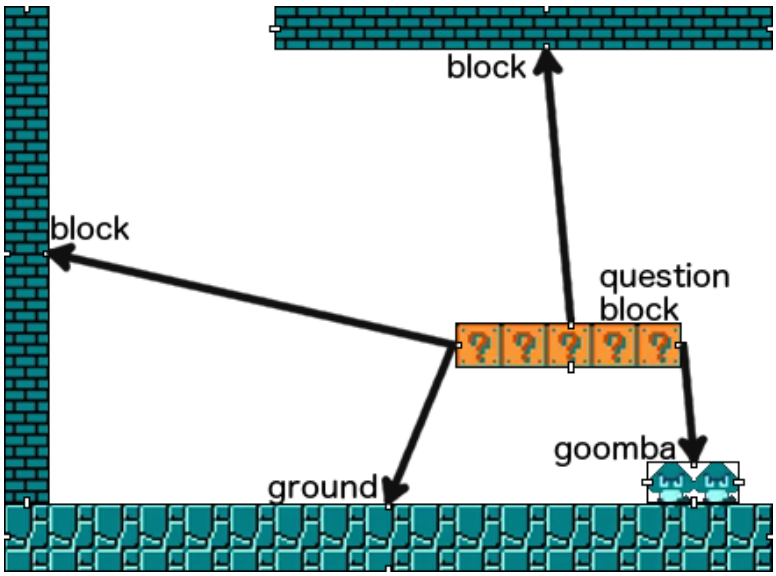

Figure 3.3: Example of a *D* Node, the set of relationships between shapes.

Shapes are built by connecting all adjacent sprites of the same type (e.g. ground, block, coin). Therefore for some types, many of the G nodes are identical, while for others the shapes represent large, irregular patterns of sprites. The D node represents the set of all relative positional differences between a given G node and all other G nodes in its level chunk. You can see a visual example of this in Figure 3.3. The D node in this case is the set of vectors capturing relative orientation and direction between the question block shape and all other G nodes in the chunk (two "block" shapes, one "goomba" shape, and one "ground" shape). The N node represents the number of individual atomic sprite values in a particular level chunk. In the case of Figure 3.3 there are two goombas, seventeen ground sprites, etc.

The first latent variable is the S node, it represents "styles" of sprite shapes. These styles vary in geometry and/or in relative positions. The system derives both the value and number of S nodes by clustering pairs of G and D nodes with K-means with k estimated according to the distortion ratio. Each pair is a G node and its corresponding D node representing all connections from that G node to all others in its level chunk. The distance function required by K-means clustering weighs the G and D parts evenly with G nodes represented

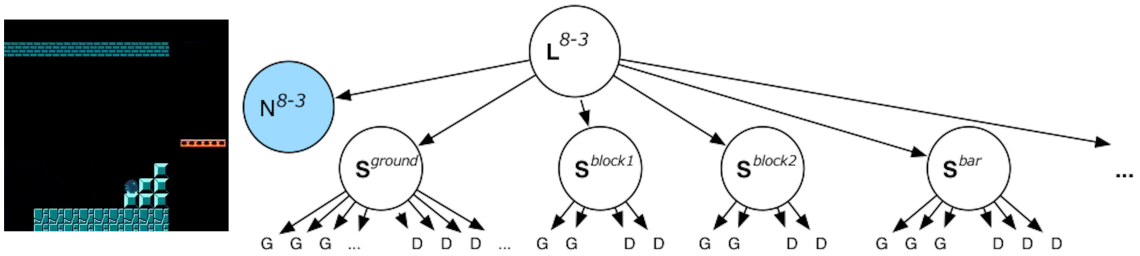

Figure 3.4: Visualization of a final *L* Node and a corresponding frame.

as binary matrices undergoing matrix subtraction and D nodes compared using Hellman's metric. Clustering with k estimation means the number of S nodes is learned automatically, with the system optimizing to best explain the variance in shapes. With a learned S node one can determine the probability of an S node shape at a given relative distance. More formally: $P(g_{s_1}, r_d | g_{s_2})$ or the probability of a G node from *within* a particular S node, given a relative distance to a second G node. For example, goomba shapes have a high probability of co-occurring with ground shapes at the same relative position as in Figure 3.3 (goombas tend to be on the ground).

The L Node represents a specific style of level chunk, the intuition behind this is that it is composed of the different styles of sprite shapes (S nodes) and the different level chunks that can be built with those shapes (N nodes). The system represents this as a clustering problem, this time of S and N nodes. Each S node tracks the N node values that arose from the same chunk as its G and D nodes. Essentially, each S node knows the level chunks from the original Mario that represent its "style" of shape. The distance metric for this stage is made of two parts. The first part is a normalized value representing the relative size of the disjoint set of sprites that co-occur with the S node's style of shape. The second part is a normalized value representing the size of the disjoint set between the N nodes that each S node tracks. This process typically leads to multiple L nodes for a single category of level chunk, in particular it tends to do a good job separating out the noisy level chunks that arise from using computer vision techniques.

Figure 3.4 visually represents a final learned L Node and all of its children. Notice the

multiple S nodes of the "block" type, with one "ground" S node. To create a new level chunk in the "style" of this L node its simply a matter of choosing an N node value and the set of S nodes values to constitute it, placing individual shapes in order to maximize their pairwise probabilities. The next section covers this process in more detail.

3.4 Generation

The system generates novel levels in the style of its training data by first instantiating a level plan based on learned transitions of level chunk categories, and then by instantiating novel level chunks to fill in this plan. This process is analogous to Joris Dormans' theory of "mission" and "space" generation [154]. The level chunk category transitions are derived from the actual gameplay video levels, using the "vocabulary" of learned level chunk categories. These transitions are then used to construct a graph representing the space of possible level designs where nodes are level chunk categories and edges represent the probability of transitioning from one level chunk category to another. I call this a *level graph*. A probabilistic walk of the graph can then create a level plan, a sequence of level chunk categories. Level chunks can be generated from the probabilistic model of shape distributions learned for each category to fill in this plan. The final level can be imported into a game engine, such as the Infinite Mario engine and played [155].

3.4.1 Generating Level Plans

To build the level graph the system utilizes the learned level chunk categories to represent the levels from the input videos. During the process of defining level chunks the system automatically discovers the end points of levels, thereby allowing the system to represent levels as sequences of level chunks. From these sequences the system learns a model of transitions between level chunk categories.

This system uses the fuzzy merge algorithm [156] to combine the individual sequences of level chunks identified when parsing gameplay video into a probabilistic, directed graph

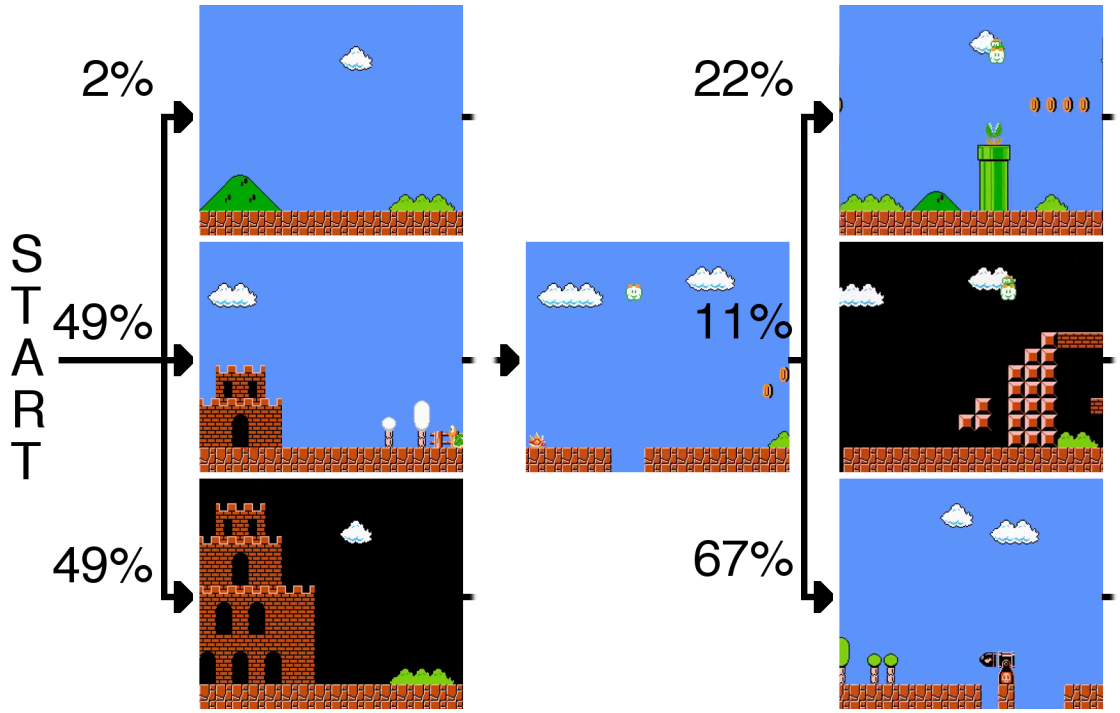

Figure 3.5: Visualization of a level graph for only overworld levels.

of level chunk category transitions. This final representation bares a strong resemblance to plot graphs [157], which have been shown to be a flexible representation for encoding long sequences of information where absolute ordering is important.

To create the final level graph, each sequence of level chunks is transformed into a linear, directed graph with each level chunk becoming a node. Each node holds its level chunk category, a normalized value representing its position in the level, and its normalized interaction time. Each node is considered in sequence, searching the entirety of the level graph for the best merge point according to a fuzzy definition of equivalence. In this case two nodes are considered equivalent if they have the same category, and their normalized position and interaction time values differ by less than 0.05. When a node is merged, it retains all outward edges, and the weight on each edge is incremented if it matches one of the outward edges in the matched node. If no merge can be found, a node is simply added to the model with the single edge to its prior node in the sequence. When this process is complete the final level graph can be used to generate each of the original level chunk

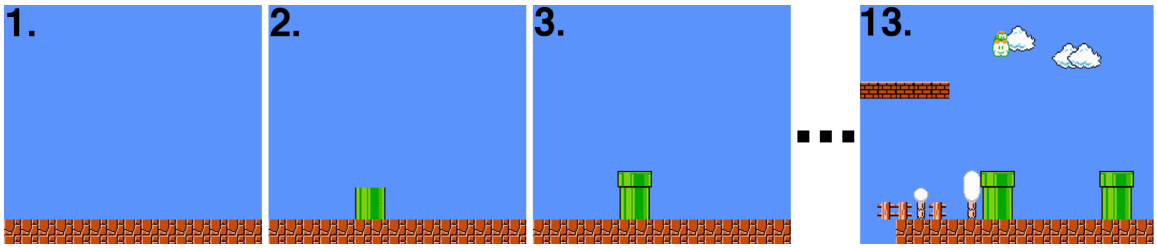

Figure 3.6: Visualization of the level chunk generation process.

category sequences and novel level chunk category sequences based on taking alternate routes. Figure 3.5 visualizes sections of the level graph learned for "overworld" levels, with each node represented by the median level chunk in that node. The fuzzy merge algorithm does correctly learn that these levels almost always start with either a big or small castle.

With the level graph, the system is able to construct a "level plan" via a probabilistic walk until it hits a dead-end, a node without any outgoing edges. The chunks of the level plan are then instantiated by generating new level chunks of the types specified by the plan.

3.4.2 Generating Level Chunks

The generation of level chunks takes place for each L node individually. The process is a simple greedy search algorithm, maximizing the following scoring function:

$$1/N * \sum_{i=1}^{N} \sum_{j=1}^{N} p(g_{si}|g_{sj}, g_{si} - g_{sj}) \tag{3.2}$$

Where N is equal to the current number of shapes in a level chunk, g_{si} is the shape of style s at the ith index, g_{sj} is the shape of style s at the jth index, and $g_{si} - g_{sj}$ is the relative position of g_{si} from g_{sj}. This is equivalent to the average of the probabilities of each shape style in terms of its relative position to every other shape style. Notably the system derives this probability by inverting the probability from the previous section with Bayes' law.

The generation algorithm begins with two things: a single shape chosen randomly from the space of possible shapes in an L node, and a random N node value to serve as an

end condition. The N node value serves as an end condition by specifying how many of each sprite type a generated chunk needs to be complete. In every step of the generation process the system creates a list of possible next shapes, and tests each, choosing the one that maximizes its scoring function. These possible next shapes are chosen according to two metrics: (1) shapes that are still needed to reach the N node value-defined end state and (2) shapes that are required given a shape already in the level chunk. The system defines a shape to require another shape if $p(s_1|s_2) > 0.95$, in other words if the shape styles co-occur more than 95% of the time. I visualize this process in Figure 3.6, starting with a G node "ground" shape and an N node value set (ground=15, pipeBody=2, pipeTop=1, cloud=2, fence=3, block=10, smallSnowTree=1, and tallSnowTree=1). The shapes not included in the starting N node come about due to being "required" for some added shape (such as the "lakitu" floating enemy).

3.5 Evaluation

I evaluated this approach with an online user study. The goal of the study was to determine if the model accurately learned to model the level design style of Super Mario Bros., which would be necessary to ensure a sufficient initial knowledge base for the video game invention problem. I hypothesized that this system would out-perform other PCGML level generators on both quantitative and qualitative metrics of style due to the probabilistic graphical model encoding the probabilities of local level structure.

3.5.1 Experimental Setup

I chose to compare the generator against two other recent PCGML approaches: the Snodgrass and Ontañón, and the Dahlskog and Togelius generators [77, 150]. I did not include the Summerville and Mateas generator and other more recent generations as they were incomplete prior to the human subjects study [73, 74, 76].

I acquired five levels for each generator to use in the user study. For the Dahlskog

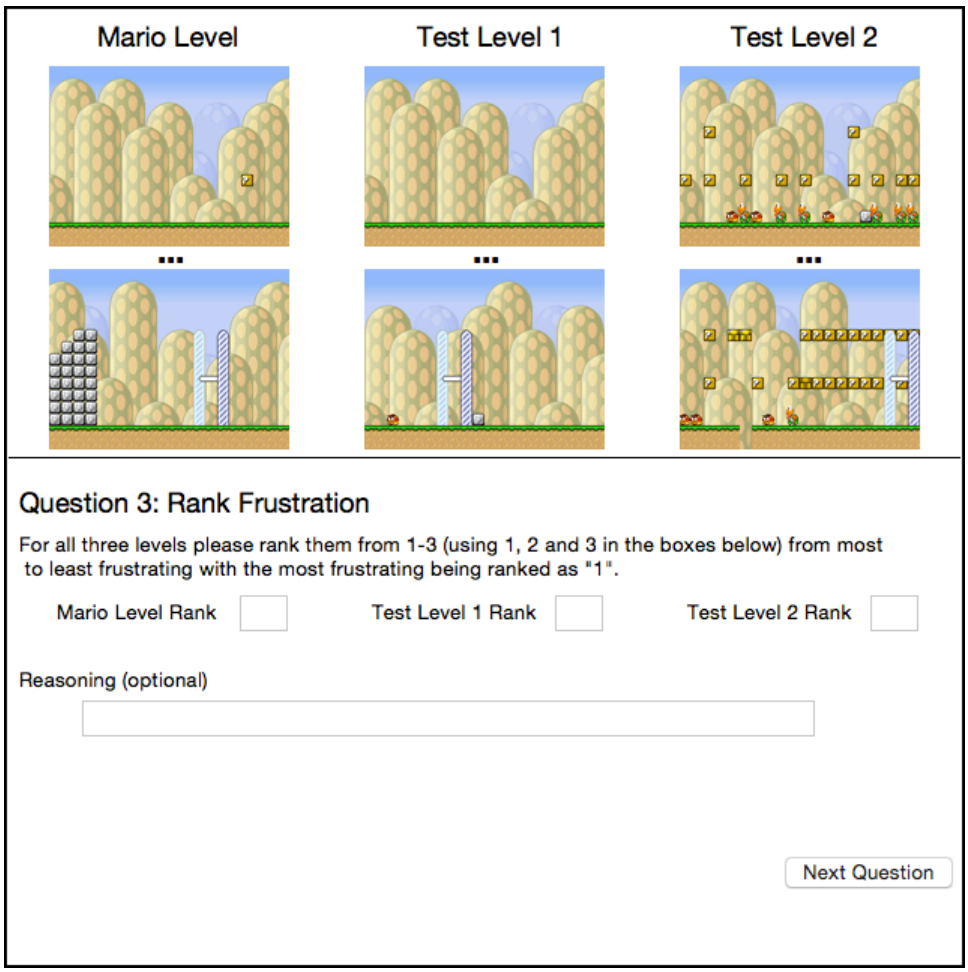

Figure 3.7: A screenshot for the Frustration question.

and Togelius generator I transcribed five randomly selected levels from the nine levels published in [77]. I generated levels from the best-performing version of the Snodgrass generator until it generated five playable levels. The Snodgrass, and Dahlskog generators were both trained on only the "overworld" levels, and so I modified my generator to only produce "overworld" levels via constraining which level chunk categories it could learn L nodes from. To deal with the fact that each L node can generate a varied number of novel level chunks (from 900 to over 400,000), I restricted each L node to only generate the top one-hundred most probable level chunks. Both other generators required a specified level length, while this approach creates levels of arbitrary length. I therefore generated one-hundred levels and selected the five playable levels closest to the specified length.

Playability was determined by a greedy-path planning agent. For the study I made use of the Infinite Mario engine [155] as the Snodgrass levels were made for this engine. I then translated both the Dahslkog levels and the levels from my approach automatically by hand-authoring the sprite mappings between Super Mario Bros. and the study engine.

3.5.2 Methodology

The study required participants to download an application, which lead them through the study. Participants played through three levels and answered ten questions: six questions involved ranking the levels they played and four collected demographic information. The first level participants played was Super Mario Bros. Level 1-1, translated into the study engine, in order to establish a familiarity with typical Mario levels. Participants were informed that this was an original Mario level. After this point each participant played two levels, one level selected randomly from the set of generated levels and one level selected randomly from one of the two other generators. The study therefore had four categories to which participants were randomly assigned, based on which generators' levels they played and in what order.

I chose to target an online population as opposed to a more specialized population like level designers for two major reasons. First, an online population interested in playing platformer game levels would likely be more similar to the population that might play the final invented games from the full system. Second, the number of individuals needed to run statistical tests comparing the three approaches was significantly larger than the pool of expert level designers I could reasonably expect to take part in a study.

The six ranking questions started with asking the participant to rank the two AI levels according to which was the "most like a level from a Mario game", which I intended as a measure of how stylistically similar the level was to ones from a true Mario game. The other five questions asked participants to rank all three played levels according to the following: Fun, Frustration, Challenge, Design, and Creativity. These types of question

appear commonly in game surveys [158, 159]. Each question was on its own screen of the study, with the top section showing the very beginning and end of each level the participant played. Figure 3.7 includes an example for the third question asking participants to rank how frustrating each level was. All other questions looked exactly like this, only changing which experiential feature it asked participants to rank.

I chose not to include the entirety of the level as I wanted participants to make their ranking based on their experience, not on an image representation. After the ranking questions, each participant was asked four demographic questions, which were all Likert-scale style questions. The first three questions were to determine how long ago the participant had played three categories of games: Super Mario Bros., any platformer Mario game, and any platformer game. The last asked how frequently the participants played games in general. I included these questions as I suspected that experience with related games might impact a subject's expectations in terms of the generated levels.

3.5.3 Study Results and Discussion

I recruited seventy-three participants over social media to take part in the study. Participants had to download an application to run the study and e-mail back a zip containing their answers and log files of their playthrough of each level. While non-ideal this was a workaround that allowed me to collect player telemetry data given that the Infinite Mario Engine is written in Java and so could not be easily hosted online. Due to random assignment I had sixteen individuals in each of the four categories, I therefore made use of sixty-four randomly selected participants' results for statistical tests.

For the first question on how "Mario-like" levels were, I ran the paired Mann-Whitney U test between the two artificial levels. For all other questions I ran Friedman's test between all three level ranking distributions as ranking values are by their nature paired and interrelated. When necessary I made use of the paired Mann-Whitney U tests to ensure individual pairs of generator's rank distributions differed significantly. I split the results

Table 3.1: The results of this approach compared to Snodgrass.

	Mario	Ours	Snodgrass	p-value
Mario-like	N/A	1	2	0.0174
Fun	1	2	3	3.02e-5
Frustration	3	2	1.5	1.06e-11
Challenge	3	2	2	0.6976
Design	1	2	3	2.31e-7
Deaths	0	2	3	2.25e-9

Table 3.2: The results of this approach compared to Dahlskog.

	Mario	Ours	Dahlskog	p-value
Mario-like	N/A	1	2	0.0383
Fun	1	2	2	0.3147
Frustration	3	2	1	3.58e-7
Challenge	3	2	1	2.31e-4
Design	1	2	3	2.21e-5
Deaths	0	2	3	2.42e-5

based on whether participants experienced a Snodgrass or Dahlskog level and include them in Table 3.1 and 3.2 respectively. I include the median rank for each generator on each question with 1 as the best and 3 as the worst (ranking it first and third), and the p-value between the generators' rank distributions. I left out Creativity, as the distribution was uniform, likely stemming from inconsistencies in what participants thought it meant for something to be evaluated as creative. I also include the median number deaths per level for each generator and the p-value from running the one way repeated measures ANOVA on each generator's death distributions. This was an objective measure of how many deaths it took a player to get through a level. Note that there was a maximum of three deaths to each level, after which players would simply fly across the top of a level.

Across all questions the rankings differed significantly between generators, except in the case of Challenge, Fun, and Creativity. These results reflect favorably on my hypothesis that my approach better encapsulated the "style" of Super Mario Bros. levels, in particular that my generator was ranked more highly than both other generators on Mario-likeness and that its Design was ranked second to Level 1-1.

Figure 3.8: The beginnings of three representative levels from each generator. Top: my technique, Middle: Snodgrass, and Bottom: Dahlskog.

I found no correlation between any of the rankings and the order in which they were presented. I did find a number of correlations between the demographic information and the rankings using Spearman's Rank-Order Correlation. For participants who played Dahlskog levels, I found a strong positive correlation between the time since the participant last played a Mario platformer and how highly they ranked the Dahlskog generator on "Fun" ($r_s = 0.54, p = 0.00147$). This suggests a reason why the Dahlskog generator and my own did not receive significantly different "Fun" rankings, given that individuals who played Mario platformers less frequently ranked the Dahlskog levels more highly. I conclude that Dahlskog levels were fun in a different way to typical Mario platformers. This matches my intuition. As seen in Figure 3.8, Dahlskog levels contain more enemies and require more precision jumps than typical Mario levels. In addition I found moderate correlations between how long since the participant had played any platformer and the number of times they died, and the former and the challenge ranking.

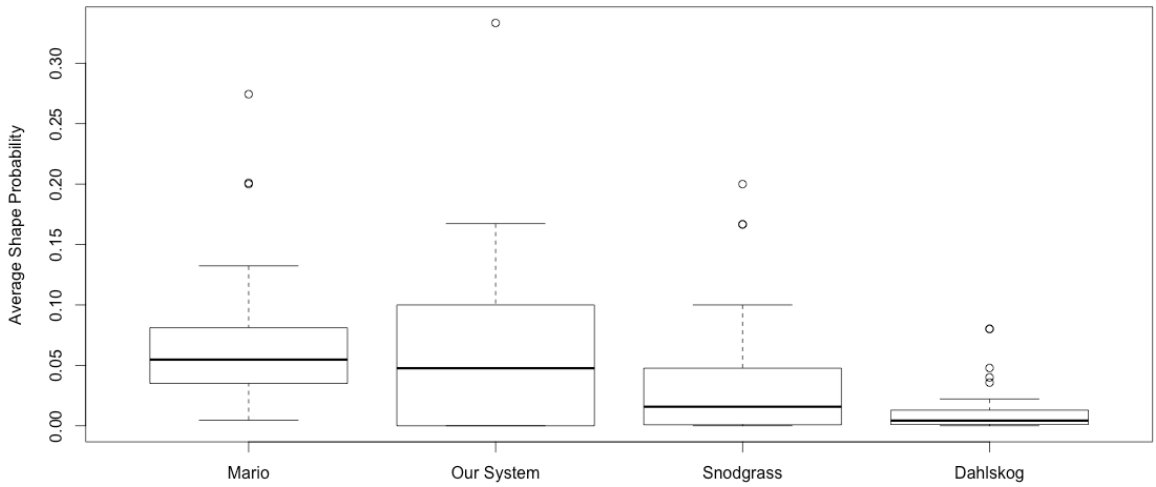

Figure 3.9: Each generator's boxplot of level chunk probabilities for the model evaluation.

3.5.4 Model Evaluation

To confirm that the model actually encodes "style" knowledge correctly I analyzed each set of levels according to the scoring metric presented in Formula 3.2. This metric captures how likely the model finds a distribution of level elements, essentially how likely the level is a Super Mario Bros. level. Given that the metric is meant for level chunks I converted each level into chunks based on screen distance, and matched each chunk to the closest level chunk from my collected training data to determine what L node to use. I present the model's evaluation of each chunk of each level in Figure 3.9. The y-axis captures the average shape probability, essentially the probability of seeing the relative shape structure (placement of game entities) for each level chunk. "Mario" in this case indicates the evaluated probabilities of the chunks extracted from the true Mario level, Level 1-1 of Super Mario Bros., while our system indicates the evaluated probabilities of my approach's levels, and so forth. Thus, the model recognizes the distribution of elements in its generated levels as being more likely than those of the other two generators. At this level of abstraction there's a clear connection between the system's evaluation of these levels and the aggregate human rankings. This suggests that the model's understanding of level design matches this aggregate human understanding. However, this scoring can be employed to derive an ap-

Category	r_s	p
Style	0.6115095	2.2e-16
Design	0.51948	2.2e-16
Fun	0.2729658	3.745e-5
Frustration	-0.4393904	6.79e-12
Challenge	-0.387222	2.351e-09
Creativity	-0.1559725	0.02007

Table 3.3: The results of comparing this system's rankings and participant rankings per question.

proximation of the model's ranking for each level, which I can then use to directly compare the human and system rankings. If these rankings correlate, that indicates that the model's evaluation and the human evaluations are likely based on similar principles. This would be ideal, given that humans are still considered the gold standard when it comes to video game level evaluation.

I employed the total ranking of levels by the model in Figure 3.9 to derive individual rankings for triplets of levels and compared these to the seventy-three human rankings. I derived the model's rankings by looking at the median value for the three levels with which each human ranking was associated. I compared the participant rankings and the model's approximated rankings with Spearman's Rank-Order Correlation. Table **??** summarizes the results with the corresponding p-values. The strongest correlation present is for the style and design rankings, which provides strong evidence that the model captures stylistic information. The other correlations can be explained as a side-effect of the model training on the well-designed Super Mario Bros. levels. The very weak correlation between the creativity rankings and this system's rankings is likely due to the lack of a strong cultural definition of creativity in video game levels. This lead to the uniform human creativity rankings and therefore the weak correlation.

3.6 Conclusions

In this chapter I present a probabilistic model of component-based video game level structure that can be use to generate and evaluate video game levels. The model requires no

manual annotation and can extract all knowledge in an unsupervised fashion from gameplay video. Thus I argue this represents the knowledge of the system, instead of my own personal knowledge baked into the system. Through a user study, I found strong evidence that the model captures style and underlying design of an exemplar set better than the current state of the art.

With the work presented in this chapter I can take a gameplay video and from it learn a model of level structure for games where the camera is locked to a single player avatar. This is helpful in representing one of the two minimal types of knowledge needed to represent a video game: level design knowledge. This chapter demonstrates that this approach gives us a strong basis for representing this aspect of video games, but it is insufficient. I still need to demonstrate the ability to learn a representation of a games rules from gameplay video as well. I present my approach to this problem in the next chapter.

At the end of this chapter one might ask whether I've solved a subproblem for the video game invention problem, the video game *level* invention problem. The focus of this chapter was in describing a process to learn a level design model that accurately reflected the level design structure in the original game. Thus, while even though I evaluated this model in terms of new levels that it output, these levels cannot be considered creative or truly novel. All of the knowledge the system employs to construct these levels comes directly from its input knowledge base, which goes against the definition of an invention problem. Thus, if I am truly to solve the video game level invention problem I must output levels that do not simply reuse the same local relationships observed in the original game. I demonstrate this in Chapter 5.

CHAPTER 4
GAME ENGINE LEARNING

In this chapter I address the problem of learning games rules from gameplay video. This problem can be understood as a subproblem of *automated game understanding*. Automated game understanding is the field of work devoted to applying artificial intelligence to derive knowledge about video game systems for the purposes of game play, design, or critique. I define *automatic game understanding* as the problem of developing formalized, complete models of the underlying processes in games. To this point its greatest success has been in the field of automated game playing. From retro Atari games [110] to board games such as Go [12] there is an understanding that playing games represents fundamental AI research. However I argue that games can offer a testbed to fundamental research beyond game playing.

Outside of the mass of work on automated game playing and some work on automated game evaluation [160, 161, 162, 163, 164] there has been little effort to automatically understand the systems that power games. However, this is an important area for research given it involves learning a model of a game's simplified physics, which could potentially scale to more realistic domains, paralleling work in automated game playing.

In this chapter, I describe a game engine search algorithm capable of learning a game engine from gameplay data. I define a *game mechanic* as a discrete rule that maps a cause to an effect (e.g. if the player is falling and hits the ground then the player stops falling). I define a *game engine* in this work as the complete set of a game's mechanics. The algorithm functions by scanning through the output of the game engine, represented as video, and iteratively improving an approximated game engine to minimize errors through greedy search. To my knowledge this represents the first approach capable of deriving an approximation of a game engine (a simulator of a specific game) only from output from the original game

engine (gameplay video). As a test domain I once again employ gameplay video of Super Mario Bros., and present evidence that this technique outputs a learned engine similar to the true game engine.

This approach allows me to represent the second type of knowledge (game rules) needed to derive the minimal representation of a game from gameplay video. Thus I need it to derive an accurate set of game rules, which can then be combined with level design knowledge derived from the approach in the prior chapter to represent whole video games. Outside of the video game invention problem I anticipate this technique can aide in applications for automated game playing, explainable AI, and gameplay transfer.

The remainder of this chapter is organized as follows. I start by providing some background on automated game playing and understanding. Section 4.2 presents the total pipeline, the proposed algorithm game engine search, and its current limitations. Section 4.3 presents the experimental evaluations and their results.

4.1 Background

In this section I situate the work presented in this chapter in terms of the most relevant prior work. *Automated understanding*, also known as common sense learning [113] or hypothesis generation [114], is the problem of taking a sequence of events, building some model that explains the events, and using the model to predict future events. An alternate approach to automated understanding common to reinforcement learning techniques is *forward model learning* [118]. In forward model learning a transition function is learned that allows an agent to make predictions in a simplified state space from sequences of state changes. Ersen and Sariel [122] stands as the most similar prior work, deriving a model of the environment for a puzzle game composed of formal logic rules based on hand-authored sequences of events. This work differs from typical forward model techniques as it learns from and makes predictions in a pixel-level state space instead of a hand-authored or learned approximation of the true state space. Summerville et al. attempt to model hy-

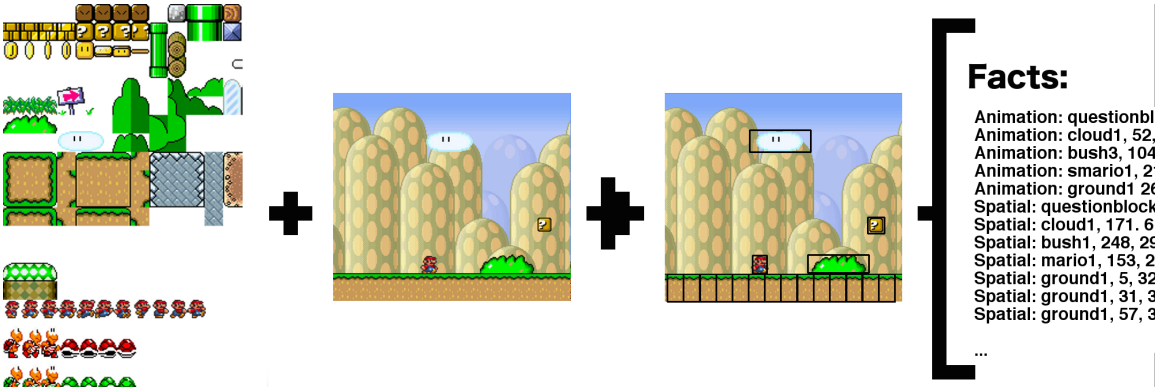

Figure 4.1: Visualization of the frame parsing process in the Infinite Mario engine . A frame is parsed to locate spritesheet elements in a frame, which then is translated into a list of facts.

brid automata by directly observing Super Mario Bros., but only model the player character [101].

4.2 System Overview

The goal of this work is to develop a computational system capable of learning a game engine, the backend set of rules that runs a game, from input gameplay video. First, similarly to the prior chapter, the system scans each input video frame to determine the set of objects present in each frame. Second, a greedy matching algorithm runs across pairs of adjacent frames to determine how objects change between frames. Lastly the system parses each frame of the gameplay video and runs an engine search algorithm if the true next frame differs from the predicted next frame by more than some set threshold.

4.2.1 Parsing Frames

I begin by supplying the system with two things: a set of videos to learn from and a sprite palette as seen on the left of Figure 4.1, using the same input as in the prior chapter. I employ Infinite Mario purely for the purposes of this figure, for the actual input I use true Super Mario Bros. gameplay video. Once again the system makes use of OpenCV [152], an open-source machine vision toolkit, to determine the number and placement of sprites in each frame of the video. All together this transforms the initial pixel input into a set of

sprites and their spatial positions in each frame.

At this point this process diverges from the prior chapter. Given a sequence of frames, where each frame is defined as the set of sprites and their locations, the system runs a simple greedy matching algorithm to match each sprite to its closest neighbor (both visually and spatially) in each adjacent frame. In the case where there are an unequal number of sprites it creates "empty" sprites to match the remainders. This will occur in the cases where a sprite is created or destroyed (e.g. when Mario shoots a fireball a sprite is created and when Mario jumps on an enemy a sprite is destroyed).

The final step of parsing each frame alters the representation from sprites to a set of facts or percepts. Each "type" of fact requires a hand-authored function to derive it from an input frame. Below I list the set of fact types and the means by which they are derived.

- *Animation:* The animation fact is simply a list of each sprite image according to its original filename, width, and height (e.g. if an image "mario1.png" of size [26px, 26px] was found at position 0,0 that would create the Animation fact $\langle mario1, 26, 26\rangle$).

- *Spatial:* The spatial fact is the sprite's location in the frame, listed as the sprite's filename and x and y position in the frame.

- *RelationshipX:* The relationship in the x-dimension of each pair of sprites in the initial frame, of the form (sprite1's filename, sprite1's closest edge to sprite2, distance in pixels, sprite 2's filename, sprite 2's closest edge to sprite1). This condition allows the system to learn collision rules. For example, Mario's x-velocity changes from positive to zero when Mario's right side hits some other sprite's left side.

- *RelationshipY:* The exact same as the above condition but for the y-dimension. This allows the system to learn collision rules in the y-dimension. Such as that Mario stops falling when his feet (bottom of his sprite) hit the top of some other sprite.

- *VelocityX:* This fact captures information about a sprite's velocity in the x-dimension and is derived according to the greedy matching to the next frame's sprite. For example if Mario is at position [0,0] in frame 1 and [10,0] in frame 2, then frame 1 will include the fact *VelocityX*: $\langle mario, 10 \rangle$.

- *VelocityY:* This fact type captures the same information as the prior type but for the y-dimension.

- *CameraX:* This is a unique fact type that simply stores a value representing how far along the camera is in the level. I included this as original attempts at this approach had an issue that when the camera moved in one direction, all stationary objects would appear (according to the other facts) to move in the opposite direction. I derive this fact by looking at the average shift in pixels of each sprite from frame to frame, therefore avoiding the issue of sliding in the opposite direction.

- *CameraY:* The same as the above CameraX but for if the camera moves along the y-axis. Notably this is not utilized in the Super Mario Bros. domain, but is helpful for learning the game engines of other games.

Notably each fact can be linked back to the characteristics of a sprite that it arose from. In other words if the spatial fact $\langle mario1, 0, 0 \rangle$ changes to $\langle mario1, 100, 100 \rangle$ the system can move the sprite mario1 to position [100, 100]. This concept is the primary method by which rules (represented in the system by changes in facts) act upon a frame.

4.2.2 Engine Learning

The engine learning approach seeks to derive a game engine that can predict the changes observed in the set of parsed frames derived in the prior section. A game engine, as defined in this approach is a set of rules where each rule is a single IF-THEN production with the If represented by a set of conditional facts and the Then representing a change as a pair of facts. For example, a rule might change a *VelocityX* fact from $\langle mario1, 0 \rangle$ to $\langle mario1, 5 \rangle$

for a given frame (THEN) given the set of facts that make up its conditions are present in that frame (the IF is satisfied).

At a high level, the game engine learning approach scans through the sequence of parsed frames and begins a search for a set of rules that explains any sufficient difference between the predicted and actual frame. If a game engine is found that reduces the difference to some threshold, then the scan begins again from the first frame of the video to ensure that this new engine has not lost the ability to accurately predict prior frames. I express the algorithm to scan through the frames in Algorithm 1 and the engine search algorithm in Algorithm 2. Notably this process begins with an empty engine that just returns whatever is passed into it outside of the behavior of the camera and the impact of velocity on sprite position.

Algorithm 1 gives the simple algorithm that scans through the sequence of frames to identify opportunities to learn for the engine search algorithm, Algorithm 2. The distance function on line five represents a pixel-by-pixel distance function between a ground truth frame ($i + 1$) and a predicted frame (derived from running the current frame cF through the current engine e) and counts the normalized number of pixels that do not match (0 if a perfect match, 1 otherwise). I choose to employ pixel distance rather than a difference in terms of each frame's set of facts as I ultimately care about the final engine being able to produce the same set of frames, not a list of facts. The $Predict$ function on line seven returns the closest frame to the ground truth frame ($i + 1$) given the current frame cF and engine e. Multiple frames can be produced by a given engine for a given frame due to the possibility of "control" rules (e.g. the player choosing to press left, right, etc), which I discuss in more detail below. I make use of a predicted frame rather than setting the current frame to the previous ground truth frame in order to build a more general game engine, rather than one that only explains frame-to-frame changes.

Algorithm 2 gives the engine search algorithm, representing the bulk of this approach. It can be understood as a greedy search to find a set of rules that creates a predicted frame

Algorithm 1: frame scan

input : A sequence of parsed frames of size f, and threshold θ
output: A game engine

```
1  e ← new Engine();
2  cF ←frames [0];
3  while i ← 1 to f do
4      Check if this engine predicts within the threshold;
5      frameDist ←Distance(e, cF, i + 1);
6      if frameDist < θ then
7          cF ←Predict (e, cF, i + 1);
8          continue;
9      Update engine and start parse over;
10     e ← EngineSearch(e, cF, i + 1);
11     i ← 1;
12     cF ←frames [0];
```

within some threshold of the actual ground truth frame. The primary means of accomplishing this is in the generation of neighbors for a given engine (as seen in line ten). Neighbors are generated via (1) adding a new rule, (2) deleting a rule, (3) modifying a rule's set of condition facts, and (4) modifying a rule into a control rule.

There are no templates for adding new rules. Adding a rule to a game engine requires picking out a pair of facts of the same type, one from the current frame and one from the goal frame, which differ in some way (e.g. one VelocityX fact with values $\langle mario1, 0\rangle$ and one with values $\langle mario1, 5\rangle$). This pair of facts represents the change that the rule handles. All other facts from the current frame make up the initial condition set for this rule, which ensures it'll be activated on this current frame. This is initially a very specific set of facts, and will include facts that are not truly necessary to activate the rule. For example, Mario can move right as long as he is not blocked to the right but an initial rule that moves Mario right might include the condition that he has a cloud above him, a bush to his left, etc.

The second type of neighbor generation simply deletes a rule from the current engine. This can be helpful for two reasons. First, it is possible that an incorrect rule was learned at some point or one that generalized too much, at which point it should be deleted. Second,

the engine uses a total ordering of all rules with rules queried for activation in sequence, and this allows the system to alter the position of rules in this sequence by deleting them and later adding them again.

The set of conditions in an initial rule can later be minimized by the third type of neighbor, modifying a rule's set of condition facts. In this case, a rule is found that did not fire for the current frame, but whose effect would be helpful in minimizing the difference between the predicted and true next frame. The intersection of this rule's conditions and the current frame's facts is taken, and this intersection set replaces that rule's conditions. In this way the initial rule from above might remove the unnecessary conditions, being left with a condition that if Mario is blocked to his right by something then Mario's x-velocity will change to 0. If this neighbor engine decreases the pixel distance of the predicted frame from the goal frame, it will be more likely to be chosen above a neighbor that simply adds a new rule as engines are placed into the PriorityQueue ($open$) according to their pixel distance and their complexity. Complexity is defined as the number of conditions summed across all an engine's rules. This preferences smaller, more general engines.

The final type of neighbor that the system handles changes a rule into a "control" rule. This covers the case of rules that the player makes decisions upon. For example, the input that controls the player character (moving Mario left, right, and jumping). Normal rules are changed to control rules under the fourth neighbor generation strategy. This allows the system to explain instances where the environment does not change, but the player does or vice versa. For example, at the start of the game Mario moves to the right not because the environment changed, but because the player hit a button to make Mario move right.

When $Predict$ is called (either in the algorithms presented or within $Distance$), this leads to a branch of two possible frames for each control rule (e.g. the player chooses to move right or doesn't). This can lead to a large number of final predicted frames as an engine can contain many control rules that each can be either active or not. The current goal frame is therefore used in $Predict$ to select from this final set the frame that is closest

Algorithm 2: Engine Search

input : An initial engine $engine$, the current frame cF, and goal frame g
output: A new engine $finalEngine$

```
1  closed ←[];
2  open ←PriorityQueue();
3  open.push(1,engine);
4  while open is not empty do
5      node ←open.pop();
6      if node [0] < θ then
7          return node [1];
8      engine ←node;
9      closed.add(engine);
10     for Neighbor n of engine do
11         if n in closed then
12             continue;
13         d ←Distance(engine, cF, g);
14         open.push(d +engine.GetComplexity(),n);
```

to the ground truth. These rules still contain a set of condition facts that must be present for the rule to fire at all, meaning not all condition facts can be active every frame (e.g. Mario cannot jump once already in the air). While this approach could theoretically lead to an engine learning that all rules were control rules (thus they could all happen or not) this is unlikely given the search heuristic preferences simpler rulesets and because this would require two neighbor generation operators (learn a rule, then make it a control rule) instead of just one.

The frame scan, stopping and restarting for engine search, continues until it reaches the end of the sequence of parsed frames. At this point one can guarantee that the system has an engine that can predict the entire sequence of frames from the initial frame within some given threshold of accuracy. Notably this means that the final engine can only reflect changes that actually occur in the input sequence of parsed frames. In addition, the choice of threshold θ has significant impact on the final engine. A θ that is too large will lead to an engine that does not represent smaller changes, since the engine search algorithm gives

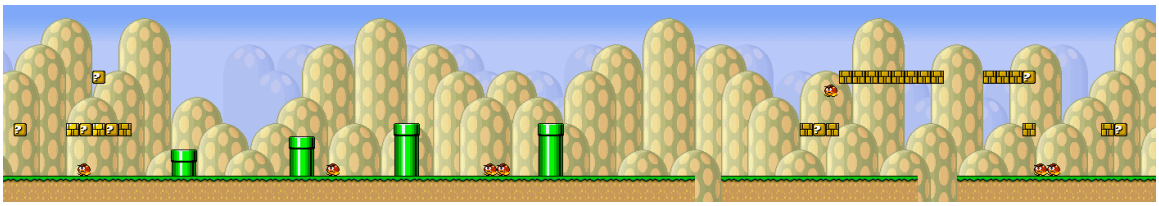

Figure 4.2: A section of Level 1-1 of Super Mario Bros. represented in the Infinite Mario engine.

preference to engines that decrease the pixel distance by large amounts. Therefore, a final engine might miss the instances that cause a sprite to begin to move or alter its position. But the smaller the value of θ the longer the algorithms will run for, which I explain further in the next section.

4.2.3 Limitations

The algorithm presented above has a number of clear drawbacks. First, it is comparatively slow depending on the threshold θ provided. While this is an offline process, meant to only run once, the running time can still be prohibitive with lower values of θ (running for up to two weeks for a single game level on a 2013 iMac).

This algorithm requires that each instance of a sequence of input data be represented as a series of facts. This requires authoring to determine the space of possible facts as discussed in Section 4.3.1. This makes the algorithm less applicable to more complex domains (e.g. real video). However, I anticipate that with a sufficient space of possible facts the technique could model any environment (whether video game or otherwise) where the majority of action occurs on screen. I identify the ability to automatically derive the space of possible facts as an important avenue for future work.

Note that the current technique only takes as training a single sequence of video, but that it can be extended to additional videos by treating the start of each new video as a new start point when the algorithm has reached the end of one video. Further, I anticipate that this would only improve the end model as a single sequence of video cannot be expected to contain all gameplay events.

4.3 Evaluation

In this section I present results from two distinct evaluations meant to demonstrate the accuracy of this system at learning an approximation of a target game engine. Given the novelty of this system I had no clear baseline to compare against. Instead I make use of two baselines to demonstrate two important characteristics of the learned engine: (1) its ability to accurately forward simulate and predict upcoming states and (2) the quality of those predictions to a game-playing task. For the first evaluation I compare the frames predicted by the learned engine against a naive baseline and a Convolutional Neural Net (CNN) constructed for frame-to-frame prediction [115]. For the second evaluation I evaluate the application of the learned engine's knowledge to training a game playing agent, compared to agents with access to no game engine or the true game engine.

For both evaluations I make use of a game engine trained on gameplay video of a single level, Level 1-1 of Super Mario Bros.. I chose to do this in order to demonstrate the approach's ability to learn from a single example. In addition I set θ to zero, to ensure the best possible output engine.

4.3.1 Frame Accuracy Evaluation

In the first evaluation I made use of a standard train-and-test procedure. I selected two examples of gameplay footage of two distinct players playing Level 1-1 of Super Mario Bros. I made use of gameplay footage that had been evaluated in prior work by Summerville et al. [106]. I drew on the two most different players and their respective gameplay videos which Summerville et al. describe as the "speedrunner" and "explorer". In this way I could ensure that the two gameplay videos represented significantly different approaches to the same level, with the "speedrunner" video taking the least time to get through Level 1-1 and the "explorer" taking the most. Thus the "speedrunner" video made up the training data for the evaluation, making the training task as difficult as possible.

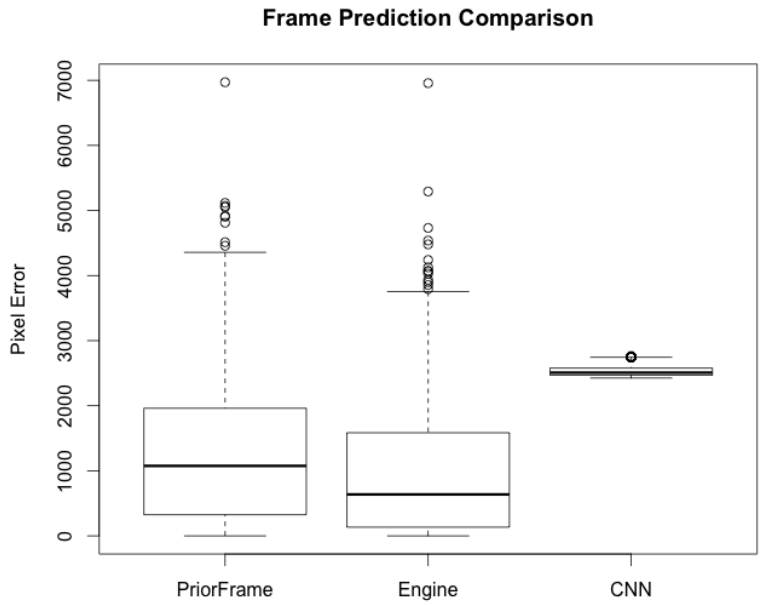

Figure 4.3: Comparison of the pixel error between the previous frame, engine's prediction, and the CNN

I drew on prior work in predicting frame transitions with convolutional neural networks (CNN) for a baseline, specifically work by Ranzato et al. [115]. I also implemented the non-time-dependent network from Vukotić et al. [116] but found it performed strictly worse, and so do not include it in the results. I made use of the same convolutional neural network architecture described in the Ranzato et al. paper, a network with four convolutional layers with 128 filters each, each making use of relu activation. For further details on the network please see the original paper. To accommodate the input to this convolutional neural net, I experimented using scaled RGB and scaled grayscale representations of the training frames. However, I found the best performance came from re-representing the frames as a one-hot encoded matrix. In other words I re-represented each frame as a 16x14x18 matrix. Where the 16x14 represent the width and height of a frame in the standard sprite-size of Super Mario Bros., and the 18 represents the 17 possible sprites that could occur in Level 1-1 of Super Mario Bros. with one index representing emptiness. This also notably treated all animating sprites as a single class. This representation gives the advantage to the CNN in terms of the size of the representation, with 4,032 possible

frames in this one-hot representation smaller by several order of magnitudes to the possible frames in the fact representation.

I trained this CNN on the same training data as the engine learning approach to better demonstrate their relative performance, meaning that the CNN was trained on only 1569 frames. I trained until the CNN's performance on the test set converged to avoid overfitting. Given that I could not directly compare the convolutional neural network error with the pixel distance function described above I instead converted all predicted frames from the learned engine and the CNN one-shot frames to a 103x90 grayscale image. I ensured to parse the 103x90 images such that the CNN could perfectly predict the correct answer (e.g. removing any part of a sprite that went outside the 16x14 tile representation of the CNN). I chose 103x90 as it was the smallest whole-number downscaling of the 412x360 frames predicted by the learned engine. I then defined a simple matrix subtraction function to compare two grayscale frames of size 104x91 pixels (varying from 0 to 9,464). I could then compare the output of the CNN to that of the learned engine across the test data by feeding each test frame and comparing the relative correctness of both approaches to the true next frame.

I present the results of this evaluation in Figure 4.3. Notably I also include the naive approach of predicting the previous frame, as an "empty" engine would simply return the prior frame, I nickname this approach "PriorFrame". While the CNN is much more consistent than either of the other two predictions, the learned engine predicts frames significantly more similar to the true frame (Wilxocon paired test, $p < 2.2e^{-16}$). Further the engines predicted frames were significantly more similar to the true frame than the "naive" assumption of predicting the prior frame (Wilxocon paired test, $p < 1.247e^{-6}$)). This provides strong evidence that the engine derives a more accurate general model of Level 1-1 of Super Mario Bros. from a single gameplay video.

4.3.2 Gameplay Learning Evaluation

From the results of the first evaluation it is possible that the system learned to make the correct predictions the wrong way. For example, teleporting Mario instead of altering his velocity to move him to the correct position. Thus, there's a need to check whether the learned game engine if functionally useful, in other words, can it be used to play the original game. However, there is no direct means of measuring whether two game engines can support the same gameplay. Directly comparing the rules of two distinct game engines does not answer this question as the same gameplay effect could be represented by several different rules (e.g. changing the velocity of a sprite versus removing a sprite and causing it to reappear somewhere else). Therefore, I instead settled on indirectly measuring the learned engine's ability to afford accurate gameplay by comparing its performance in training a game playing agent.

For a ground truth I again drew on the "Infinite Mario" engine used for the Mario level playing AI competition [165] and constructed a simple reinforcement learning agent for the engine utilizing the value iteration algorithm. I further constructed two variations on this agent. The first made use of the native forward simulation available in the "Infinite Mario" engine to construct an initial policy rather than relying on the typical approach of a uniformly random initial policy. In particular I set the initial value for each state s to $V[s] = max_a \sum_{s'} R(s, a, s')$ according to a reward function that gave infinite reward for reaching the end of the level and negative infinite reward for deaths. This can be understood as changing the value iteration problem into a simple greedy agent, and represents a theoretical "lower bound" for predicting the quality of the next state. In comparison, I constructed an agent that instead made use of forward simulation according to the learned game engine (with numerical values and names converted to the appropriate equivalent in the "Infinite Mario" engine). For each agent I defined the current state as the size of a screen and Mario's current x and y velocity. Since each of these agents in essence made use of slightly different reward functions I compare the number of iterations required for

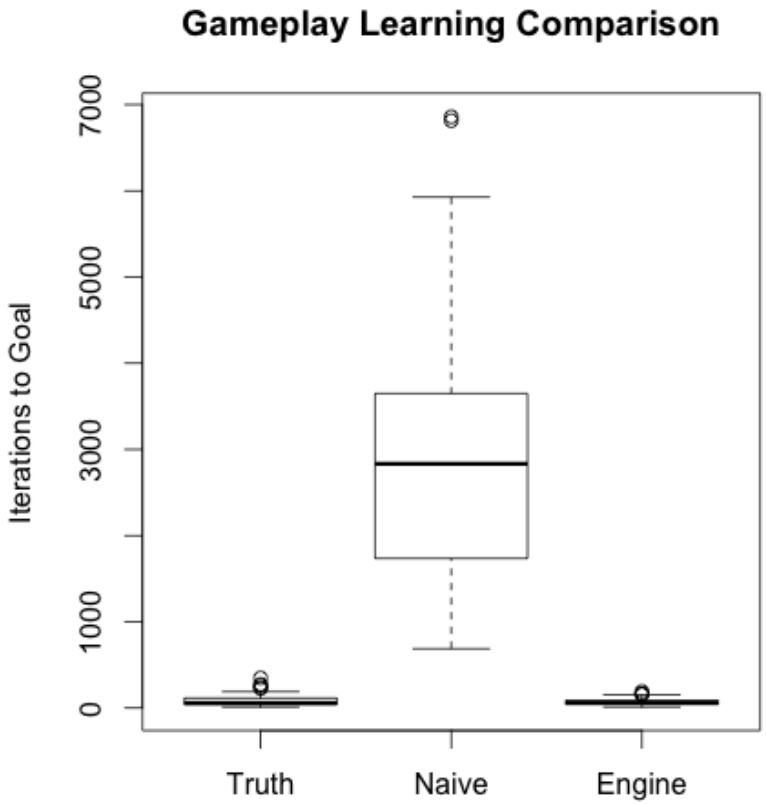

Figure 4.4: Comparison of the iterations required to reach the goal across the three agents.

each agent to reach the goal state. As a baseline I include a "Naive" or model-free approach that makes use of a standard uniformly random initial policy.

I ran 100 agents of each type over a Infinite Mario version of Super Mario Bros Level 1-1. (Figure 4.2) and compare the iteration where each first reached the goal state in Figure 4.4. As anticipated, both the true engine and learned engine out-performed the basic RL agent. Notably, in comparing the true and learned engines in terms of their impact on their respective agents, I was unable to reject the null hypothesis that the values came from the same distribution (Wilcoxon paired test $p = 0.5783$). This represents evidence that the true and learned engines represent similar enough gameplay knowledge that they are indistinguishable, at least for this task. However, given that the learned engine is only trained on Level 1-1 of Super Mario Bros. I would not anticipate the same similarity in performance for a different level, especially on that drew on different mechanics.

4.4 Conclusions

In this chapter I presented a novel technique to derive a game engine from input gameplay video. I ran two evaluations, which provide strong evidence that the learned engine represents an accurate, general model of the video game engine responsible for producing the input gameplay video. This technique relies on a relatively simple search algorithm that searches through possible sets of rules that can best predict a set of frame transitions. To my knowledge this represents the first technique to learn a game engine, a set of rules capable of simulating a game world from gameplay video. Beyond allowing me to represent the mechanical aspect of games in the video game invention problem this represents a major step forward in the field of automatic game understanding.

Combined with the results from Chapter 3, I now have the ability to learn an approximation of game rules and level design from gameplay video. Two challenges remain. First, there is still nothing novel about what I have presented, given that the work in the prior and current chapters focused on replicating existing knowledge as accurately as possible. I address this in the next two chapters in which I investigate the creation of new types of video game levels via combinational creativity. Second, I still require a means of representing both game rules and level design in the same representation, which I address in Chapter 7.

CHAPTER 5

CONCEPTUAL BLENDING OF MACHINE LEARNED MODELS

In the prior two chapters I have discussed my approaches to learn the knowledge required for a video game representation that will make up the knowledge base for the video game invention problem. By the nature of this task there has been very little novelty in the output learned models. The goal for both approaches was to instead replicate existing knowledge as much as possible, to mimic instead of to invent. I would not argue that the work I have presented so far demonstrates any real creativity. The approach I presented in Chapter 3 produced technically new levels, but these were levels that reused existing local structure taken from the input training data. Further they represented only a single, existing level class or type: above ground or "overworld" levels. To solve the invention problem for video games we would need to create new types of levels with new types of local structure. One might, for example, consider a combination of a boss or castle level and an underwater level, a kind of level that has never existed before.

In the first chapter I proposed combinational creativity as a means of creating the new knowledge required to solve invention problems. Conceptual blending has traditionally been the most popular combinational creativity approach. Conceptual blending has traditionally appeared in expert systems applications, where a human expert encodes concepts from a particular field such as architecture, engineering, or mathematics [29, 31]. Despite conceptual blending's creative potential, it has not appeared in the domain of video games to any large extent, even though games are well-suited to computational creativity research [63]. This is likely due to conceptual blending —and many other computational creativity techniques— relying on high quality knowledge bases. The quality of the knowledge base determines the quality of the blends a system is capable of constructing, meaning that a human expert often has to iterate over a knowledge base multiple times.

Conceptual blending systems take a significant amount of human effort to construct. Machine learning could in theory derive a knowledge base of machine learned models from a corpus of examples, thus reducing the requirement of human input. However, models learned from machine learning techniques tends to be noisy, full of inconsistencies and mistakes that could thwart typical approaches to conceptual blending. Notably, I am not proposing applying conceptual blending over individual training data instances, but over models trained on these instances.

In this chapter I cover an experiment applying conceptual blending to machine learned models. Specifically I draw on the machine learned models of level design discussed in Chapter 3. This experiment allows me to identify the benefits and limitations of a naive application of an existing combinational creativity approach to machine learned models. I employ conceptual blending rather than another existing combinational creativity approach due to the fact that it is the most well-researched of the existing combinational creativity approaches. In addition, this represents an important contribution to the problem of video game invention. The work in this chapter addresses a smaller or sub-invention problem: the invention of novel video game level types. I evaluate this work through a case study of levels from the original game designer of Super Mario Bros., given that these level can be understood as combinations of previously existent level types.

5.1 Background

Fauconnier and Turner [48] formalized the "four space" theory of conceptual blending. In this theory they described four spaces that make up a blend: two *input spaces* represent the unblended elements, points (i.e. features or components) from the input spaces are projected into a common *generic space* to create a mapping between elements, and these mapped points are projected into a *blend space*. In the blend space, novel structure and patterns arise from the projection of these mapped points. Thus a typical conceptual blending process begins by mapping similar components of the two inputs, and then combining

these mapped components in a final representation. However, due to the large amount of authorship required, it is unclear if the creative output of historical systems arises from conceptual blending algorithms or the creativity of a human author when encoding structures. Recently O'Donoghue et al. [49] have looked into deriving this knowledge automatically from text corpora, producing graphical representations of nodes and their verb connections. This work runs parallel to O'Donoghue et al., but in the domain of two dimensional video games levels.

Conceptual blending, based on analogy or any other mapping technique, is not commonly used in video games. Prior work has looked into knowledge intensive conceptual blending systems to create new elements of video games such as sound effects and 3D models [65, 66]. Permar and Magerko [69] presented a system to produce novel interactive narrative scripts via conceptual blending, using analogical processing. The work presented in this chapter focuses on the combination of level design models for a platformer game, a very different domain.

5.2 Conceptual Blending of Level Design Models

The levels generated from the learned probabilistic model discussed in Chapter 3 tend to resemble the original Super Mario Bros. levels, and while novel, may not be considered creative or surprising. While the model may generate a unique configuration of elements, the types of elements and their individual relationships are drawn directly from the original game. It can hardly be argued that this represents a satisfying example of video game level invention. Conceptual blending serves as a well-regarded approach to produce creative artifacts, but the learned models extracted from the gameplay videos are not well-suited to conceptual mapping due to their complexity. Instead of using these models directly, I take the common conceptual blending approach and transform the detailed model into a more abstract or generic model in order to find mappings [29]. This can be thought of as transforming these models from their input space into the generic space in the language of

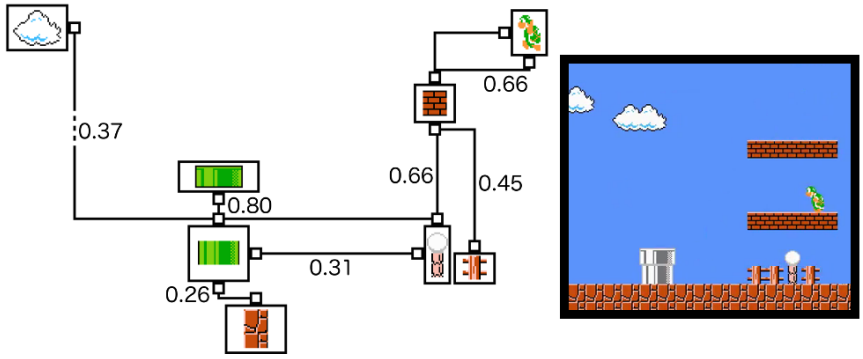

Figure 5.1: S-structure graph and an example of level chunk associated with it.

Fauconnier and Turner [48]. I call this generic model an *S-structure graph*, and define it as the set of S nodes (styles of sprite shapes in a model) and a set of edges representing probabilistic relative positions between them.

5.2.1 Deriving S-Structure Graphs

Figure 5.1 gives an example of a final S-structure graph on the left derived from an L Node trained on level chunks like those of the chunk to its right. Each box and image represents an S node, the lines between them are D node connections, vectors connecting the cardinal points of the S nodes. The D node connections also have a probability [0...1] corresponding to how likely they are to appear in a particular level chunk.

Each S Node has many more D node connections than those that appear in the S-structure graph. The D node connections I employ as edges are the minimum number of connections with the maximum probability to create a fully-connected graph. An S-structure graph is constructed by first taking all of the S nodes of a particular L node as nodes in an initially completely unconnected graph. Then the most probable D node connection across all the S nodes that most grows the maximal connected subgraph of the graph is added. When the graph is fully connected, the process stops.

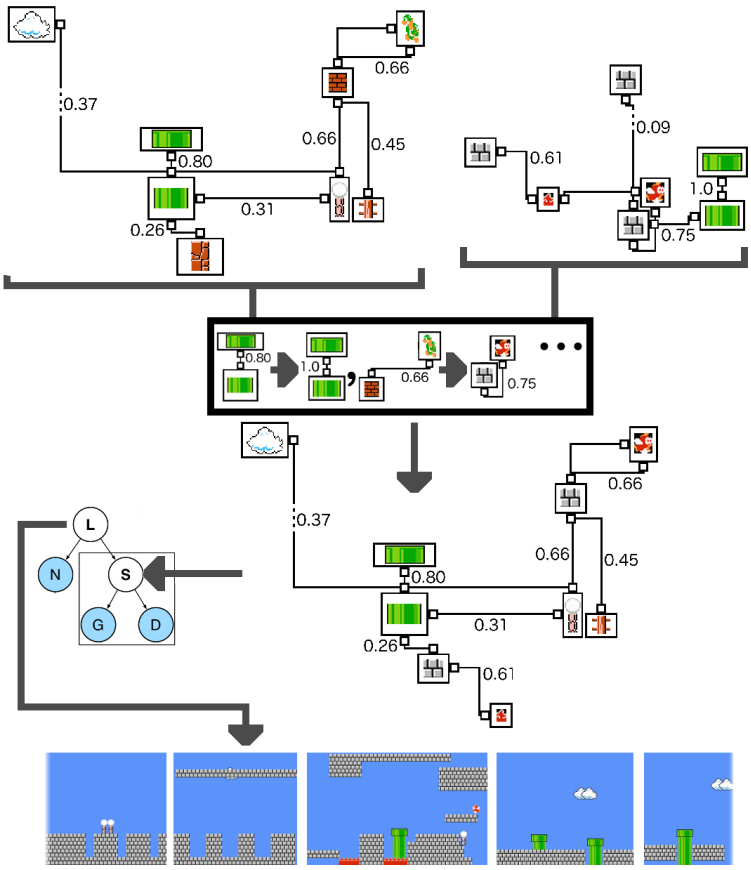

Figure 5.2: Visualization of the conceptual blending implementation.

5.2.2 Blending S-Structure Graphs

I visualize the complete blending process for S-structure graphs in Figure 5.2. At a high level the process takes two input L nodes and a final set of goal S node types (e.g. names of sprites). For example for Figure 5.2 the final set would be: cloud, pipe top, pipe bottom, dungeon block, white tree, fence, lava, and cheep cheerp (the little red fish). Notably this set of S node types must represent a subset of the total set of available S node types between the two input graphs. This set of goal S node types can be understood as a goal for the blending process as in [51]. The process ends with a final L node composed of actual S nodes whose types match this goal.

Conceptual blending systems often have a concept of a *source* space and *target* space. My approach is the same, in that one of the L nodes is considered the source and the other

L node the target. In this case this means that the source L node will end up modified to create the final blended L node.

First, these two nodes are transformed into S-structure graphs according to the process described in the prior section. This can be seen at the top of Figure 5.2 with the S-structure graph at the top on the left serving as the source and the S-structure graph on the right serving as the target. As described previously this S-structure graph representation allows me to simplify the problem of combining L nodes with conceptual blending.

With the two inputs transformed into the generic space representation of S-structure graphs the process must next construct a mapping. By mapping in this case I mean a one-directional matching, what target graph components correspond to what source graph components. Notably, there is the possibility that the same component from the target graph will be paired with multiple components in the source graph. The end goal is to map S nodes onto one another in order to hit the target set of goal S node types. However, the S nodes in the S-structure graph carry no information besides their S node type and are therefore not helpful for determining a mapping. Instead I construct an initial mapping based on the edges. Each relationship—D node connection—from the source S-structure graph is mapped to the closest relationship on the target S-structure graph. This mapping is constructed via a simple closest match, based on a function that equally weighs differences in probability with the cosine distance of the two vectors. This mapping can be seen in the middle of Figure 5.2.

This list of D node connection mappings can be transformed into a list of S node mappings via referencing the S nodes the relationship exists between. For example if an edge that connected a "goomba" S node and a "ground" S node was mapped to an edge that connected a "hammer bro" (the turtle creature in Figure 5.1) S node and a "brick" S node that would lead to two mappings: "goomba" to "hammer bro" and "ground" to "brick". Notably this process can lead to multiple mappings for each source and target node based on the number of edges connected to these nodes.

Figure 5.3: World 9-1 from the game Super Mario Bros.: Lost Levels

The system then determines what source S nodes should be swapped out for what target S nodes according to the goal set of S node types. In particular for each S node type present in the target graph but not present in the source graph the system must decide what source S node to replace (via blending) with this target S node. This is determined by finding the non-goal source S node most frequently mapped to a goal target S node, with ties broken according to similarity of edges. For example, if the desired final set was "brick, goombas, and hammer bro" then the system could accept the mapping ground to brick, but not goombas to hammer bro.

This final mapping leads to a blended S-structure graph that is then used to create the blended, final L node as can be seen in Figure 5.2. In particular each pair of S nodes is blended (combined) by taking the G and D values of the source S node in the source L node and replacing their type with the target S node type. For example, in this running example of mapping a ground S node to a brick S node, all of the shapes and relationships of the final blended S node would be that of the ground node, but instead pertaining to bricks (e.g. the same shapes and placements you'd typically see for ground sprites, but with bricks instead). Notably all of the N node values are also changed to reflect the S node mappings, so for example every time "ground" was mentioned in an N node value it would now change to "brick".

This final blended L node is capable of generating and evaluating chunks of levels according to the process described in Chapter 3. For example output level chunks can be seen at the bottom of Figure 5.2. Notably these represent entirely novel local structure, with these individual game components never co-occurring in these ways in the original Super Mario bros. games.

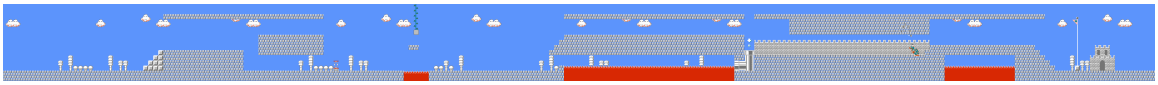

Figure 5.4: World 9-3 from the game Super Mario Bros.: Lost Levels

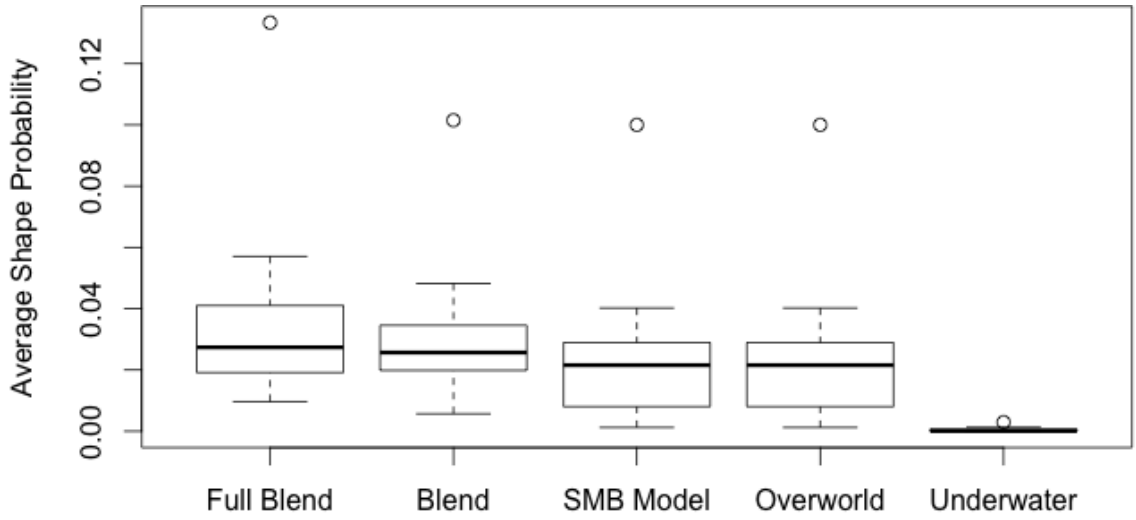

Figure 5.5: Score distributions from evaluating World 9-1

5.3 Blending Evaluation: Lost Levels

The evaluation of combinational creativity techniques is a traditionally difficult problem due to the subjective nature of combination quality. Given that the blended models are generative, I could run a human study on levels generated from these models. However, the human study from Chapter 3 demonstrated that human subjects do not tend to agree on the creativity of a level, indicating that this type of study might prove inconclusive. The learned level design models can also be used for *evaluation*, thus an alternative way to determine the quality of the blended models is to examine how well they account for human-expert levels of blended level types.

In the case of Super Mario Bros., the designer Shigeru Miyamoto designed a second game known as Super Mario Bros.: Lost Levels (Super Mario Bros. 2 in Japan) based on the original game. The game includes levels that can be understood as blends of Super Mario Bros. level types. World 9-1 (Figure 5.3) uses a combination of sprites found otherwise only separately in "underwater" and "overworld" levels. The level includes castles,

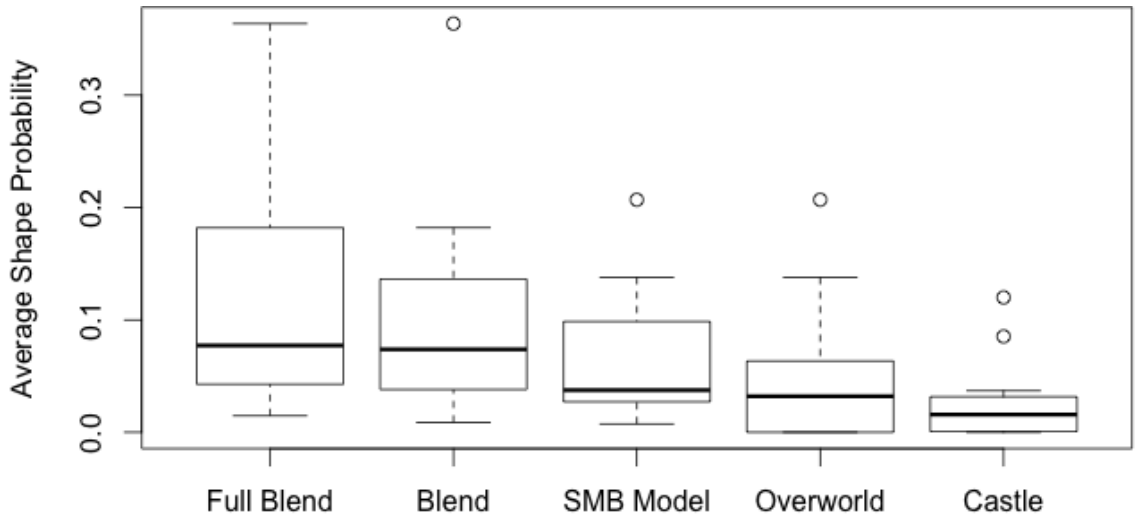

Figure 5.6: Score distributions from evaluating World 9-3

clouds, and bushes that only appear in overworld levels appearing with coral and squids. World 9-3 (Figure 5.4) on the other hand uses a combination of sprites otherwise found only separately in "castle" and "overworld" levels. The level includes elements from overworld levels alongside lava and castle walls.

Due to their "blended" nature, I hypothesize that the blending technique can create models that explain these human blends significantly more than the original, unblended models trained only on the original Super Mario Bros. levels. Essentially, because these levels represent a human example of combinational creativity there are two possible outcomes. First, if conceptual blending does a reasonable job of approximating this process then the blended model will find the local structure of these levels more probable in comparison to the unblended, original model. Second, and alternatively, the conceptual blending could lead to a blended model that does not capture the same kind of local relationships seen in these levels, and it will do no better then the original model. To evaluate these levels with the different models I used the same strategy as the model evaluation from Chapter 3, splitting each level into uniform chunks and evaluating each chunk in terms of the probability of its structure according to each model.

I created four different versions of the system to create four distinct types of level design

evaluation model:

- **SMB Model**: The Super Mario Bros. (SMB) model represents the set of L nodes learned from gameplay video of the original game.

- **Blended Model**: To construct a blended model the system first chooses what of the original L nodes to blend. The system constructs this initial set by choosing the L node that maximally explains each chunk of the blended level. The system then blends each pair of L nodes in the set as both the source and target L node using the sprites in the blended level as the goal set of S node types. This model can be thought of as an unsupervised model.

- **Level Type Model**: I constructed additional models via hand-tagging each L node with it's level type. For example, "Overworld" to represent the above ground levels, "Underwater" and "Castle". These models represent subsections of the larger SMB model. I parsed each blended level with the level type models that made up its blend. World 9-3 (Figure 5.4) was therefore parsed with the "Overworld" and "Castle" models.

- **Full Blended Model**: I constructed the largest possible blended model for each level as what I call a "full" blended model. I took all of the L nodes tagged with the two level types that makes up each blended level, and blend all of the L nodes together for all possible pairs, leading to a massive final blended model. This model serves as an upper-bound of performance for the blending technique given human knowledge of level types. This is the much more typical approach in the computational creativity field: adding semantic knowledge to a system via tagging. However it has the potential to limit the degree to which one can claim the output of the system represents its own artificial creativity, given that the content may just have been creativity tagged.

Figure 5.5 summarizes the results of the evaluation for World 9-1. While 9-1 is made up of a combination of "overworld" and "underwater" level sprites, it is much more over-

world then underwater with a 6:1 ratio of sprites in the level favoring overworld. The models reflect this, with the Underwater level type model doing very poorly at explaining the level, while the SMB and Overworld level type models behave essentially the same. Despite this low quality blend, the blended model's distribution differs significantly from the SMB Model distribution according to the paired Wilcoxon-Mann-Whitney test (p=0.03327). Thus conceptual blending lead to a model that better represented the kinds of relationships seen in World 9-1, notably without ever training on anything in World 9-1 directly (i.e. using the process described in Chapter 3). In addition the blended model and full blended model distributions do not differ significantly ($p > 0.05$), indicating that the system's choice for L nodes to blend is as good as creating all possible blended L nodes with human tagging in this case. It is worth noting that the SMB model typically finds median scores for actual Super Mario Bros. levels between 0.1 and 0.2, with the lowest median score for any level being 0.05. None of these models reaches even the lowest point, but I contend this is due to the fact that the level does not represent a strong blend.

Figure 5.6 summarizes the results of this evaluation for World 9-3. World 9-3 represents a much more even blend than World 9-1 with an overworld to castle sprite ratio of 3:1. This is reflected in the relative distributions of the Castle and Overworld level type models. Once again the blended model distribution differs significantly from the SMB Model distribution (p=0.0008308). In this case the full blended model also differs significantly from the blended model (p=0.002961). However, despite the overall higher distribution, the full blended model's median value rose only a small amount compared to the blended model's median (0.077 vs 0.074). The full blended model is also made up of over two-hundred L nodes as opposed to the blended model with only twenty-four L nodes. This suggests that my approach to conceptual blending picked out the most important L nodes to blend. Notably, both blended models' distributions fell into the range of an actual Super Mario Bros. level. I contend this is due to the level being a more even blend, thus the blended models were equivalent to models trained directly on levels like World 9-3.

Figure 5.7: A high-quality blended level according to the model targeting World 9-1.

Figure 5.8: A high-quality blended level according to the model targeting World 9-3.

From these results its clear that the blended models do a better job of explaining the structure of World 9-1 and 9-3 compared to the original models. Thus, I can reject the possibility that conceptual blending would lead to probability distributions that did an equivalent or worse job at evaluating these levels. This lends some evidence to Fauconnier and Turner's [44] theory of conceptual blending as a model for human combinational creativity. But more importantly it demonstrates positive evidence towards the utility of combining combinational creativity and machine learned models.

5.3.1 Example Output

I present a set of illustrative generated levels from this approach. To create full levels the system determines the sequence of L nodes that best explains the sequence of uniform chunks of a target level. Each L node in this sequence is then prompted to generate a novel level chunk and this sequence of generated chunks constitutes a level. Figure 5.7 and Figure 5.8 represent high-quality levels (according to the system's own evaluation) using a blending target of World 9-1 and 9-3 respectively. In comparison Figure 5.9 represents a lower quality blended level of 9-3, and Figure 5.10 represents a high-quality level generated by the *full* blend model for 9-3. The difference between the low and high scoring levels should be clear from their structure, with Figure 5.9 including individual, oddly placed

Figure 5.9: A lower quality blended level according to the model targeting World 9-3.

Figure 5.10: A high quality blended level generated by the *full* blend model targeting World 9-3.

Figure 5.11: An example of an underwater castle level created by hand-defining a set of final S nodes, specifically the Castle Block, Cheep Cheep, Squid, Coral, Bar, Question Block, Bowser, and Castle Bridge.

blocks and a floating castle. From my perspective I note a lack of difference between the blend and *full* blend models, with Figures 5.8 and 5.10 appearing very similar.

5.4 Underwater Castle Case Study

The prior section showcases levels created by the blended L nodes. However, these were constructed based on existing, human-authored levels. An alternative approach that would produce a totally novel level type would be to define a set of final S nodes without some example level, using the same process as above with this set as a goal for the blending process. This set could be constructed according to a number of strategies, but as an example I employ a hand-authored final set for a theoretical "underwater castle" level type. I include the output of this process in Figure 5.11. This represents a type of level that has never existed in any game in the Super Mario Bros. series. Thus this is a reasonable attempt at the video game level invention problem, with the caveat that the mapping was constructed by hand and that this level has not been evaluated outside of the fact that it is playable (an important element of value for video game levels) and novel.

5.5 Conclusions

In this chapter I presented an initial study of the application of combinational creativity to machine learned models and an attempt at the video game invention problem. I demonstrated via two case studies that conceptual blends of the learned level design models are

able to explain human expert blended levels, and able to explain these levels significantly better than the unblended models. This is helpful as evidence for the utility of combinational creativity when applied to machine learning and a step towards the video game invention problem.

Conceptual blending has a number of limitations. First, it historically only functions over symbolic data. While I treated the edges and nodes of the S-structure graph as symbolic values for the purposes of blending this limits the potential output of this process. Second, conceptual blending of this sort only outputs a single correct blend from any two inputs, thus why I treated each pair of L nodes as both source and target to maximize the number of output, blending L nodes. Third, I had to come up with an entirely new representation, the generic S-structure graph, in order to address the messiness of the machine learned models, which is an added burden of this approach. Taken together, these stand as barriers to the application of conceptual blending to solve the video game invention problem. In the next chapter I introduce conceptual expansion formally to address these and other limitations.

CHAPTER 6
CONCEPTUAL EXPANSION

The prior chapter introduced the application of combinational creativity to machine learned models of level design. In this chapter I present a more thorough study of three existing combinational creativity approaches in terms of their output spaces for the domain of video game level combination. While conceptual blending is included, this work differs from the prior chapter. The approach introduced in the last chapter only output one blend for each pair of inputs. In this chapter I present a more general implementation of conceptual blending to investigate its full possible output space. I do the same for amalgamation and compositional adaptation. In comparison to these approaches I introduce a novel combinational creativity approach I call *conceptual expansion*, designed to address the limitations of historic combinational creativity approaches.

Conceptual expansion structures the combination task as a function over each output component whose parameters are possible input components and filters that specify how the combination occurs. This formulation allows for much more fine-grained control over the combination process and allows conceptual expansion to handle continuous variables and messy, machine learned input. In particular, the filters allow for conceptual expansion output that ignores large portions of its input. This runs counter to many historic combinational creativity approaches, which look to minimize information loss by incorporating as much of the input as possible in the output.

The remainder of this chapter is organized as follows. First, I reintroduce and formally describe three historic, general combinational creativity approaches: amalgamation, conceptual blending, and composition adaptation. Following this I present a comparison of these approaches in terms of their output spaces and identify their limitations. I then introduce and formally define conceptual expansion. Finally I present the case study applying

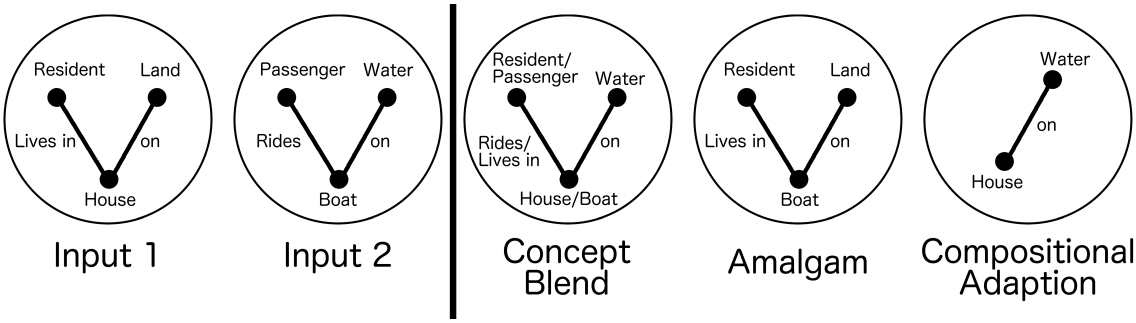

Figure 6.1: Example of three combinational creativity techniques. Two input spaces on left with example output from the three techniques on the right.

all of these approaches to the video game level invention problem, comparing them in terms of their complete output spaces.

6.1 Background

In this section I present descriptions of the three historic combinational creativity techniques I employ for comparison in this chapter: conceptual blending, amalgamation, and compositional adaptation, I chose these three techniques due to their relative domain independence. For a grounded example of these approaches in terms of their output see Figure 6.1, which was originally presented in the first chapter. Throughout this chapter I make use similar graph-based representations, which assumes individual nodes with edges connecting nodes that relate in some domain-dependent way.

At one level of abstraction all of these approaches follow the same three-step process. First, take in two graphs as input, as seen on the left side of Figure 6.1. Second, construct a mapping between elements of the two graphs. Third, run the individual algorithms that derive the final output combinations from the mapping.

The mapping typically requires some outside knowledge, either in the form of hierarchical information (e.g. houses and boats can be said to have the same parent object), a distance functions, or an authored mapping. Alternatively, mapping can make use of graph structure itself, looking for similarities in node and edge relationships between graphs, irrespective of the semantic meaning of the node and edge values. The purpose of the mapping

is to help limit the final possible combinations to those that leverage the domain-specific structure of the two inputs.

As an example, the mapping for Figure 6.1 maps *house* and *boat*, *live in* and *ride*, *passenger* and *resident*, and *land* and *water* as taken from [8]. This mapping was hand-authored, but could have arisen from the structural similarity of the two inputs (which were also hand-authored to have this structural similarity). In this way all of the output from the three different approaches reflects this semantic knowledge. It is impossible to do without this mapping completely, but one could imagine alternatives. For example, if instead all nodes were considered equivalent with one another one could end up with the final amalgams like "Water Lives in Resident on Land", "Land Rides Boat on Passenger", "House Rides Passenger on Resident", and so forth. While the first of these retains some semantic meaning (water does make up the majority of the human body), the majority of the possible amalgams would be semantically meaningless.

After a mapping has been established, each approach makes use of a unique algorithm to construct final combinations, which I discuss below. Notably there are typically many possible final combinations for each approach, depending on the two input spaces and constructed mapping. However, this is typically viewed as undesirable, with general heuristics or constraints designed to pick a single combination from this set [44, 6].

I now describe the three existing techniques I highlight in terms of the algorithmic approach each uses to construct combinations from a given mapping. As a reminder I focused on these three existing approaches for two primary reasons. First, their relative domain independence, with each of these approaches having been applied in multiple distinct domains. Second, their distinct output spaces, which I further describe later in this chapter.

6.1.1 Amalgamation

Amalgamation, the process that produces amalgams, considers mapped elements to be incapable of co-existing in a final amalgam. In the example of Figure 6.1 house and boat

could not both be present in a final amalgam. Instead amalgamation first chooses one of each of the mapped elements, then adds all of these chosen elements and as many of the non-mapped elements as possible based on node-edge relationships to the final output graph. One can understand amalgamation as most like the every day usage of the term combination, with no mixing of the individual mapped elements.

6.1.2 Conceptual Blending

Conceptual blending produces blends. As described in Chapter 2, historically conceptual blending relies on four distinct spaces: the two input spaces, a generic space, and the actual blend space. The abstract mapping stage discussed above can be understood as the mapping into a generic space. From this generic space mapping the blending algorithm derives a final mapping to apply in the blend space, which can vary from an empty mapping to the full generic space mapping. In other words, the conceptual blending algorithm can essentially unmap some of the mapped elements for its final blend space mapping. For example in Figure 6.1 the displayed output is from a blend space mapping equivalent to the general mapping described above except for the mapping of land and water as discussed in [8].

All individual mapped elements are combined into a single new blended element, such as *house/boat* and *rides/lives in*. The exact way that this blending of components occurs depends upon the particular domain, with the notion that as much as possible of each input component should be represented in the final blended component. For example in a Minsky Frame-esque series of keys and values [166], one might represent the *boat* component with the variables "Name: boat, Size: large, Steerable: true" and the *house* component with the variables "Name: house, Size: large, Dwelling: true". Then the final blended component might have the values "Name: house/boat, Size: large, Steerable: true, Dwelling: true". The final blend is composed of all of these blended elements and as many of the non-blended elements as possible given open node-edge relationships. The goal is to maximize the shared information between the blend and the two input spaces. The choice of what

blend space mappings to apply represents the majority of conceptual blending's expressive power. Typically this theoretical choice is ignored in whatever process is employed to construct blends for a particular instance of conceptual blending, as with my approach in the prior chapter.

6.1.3 Compositional Adaptation

Compositional adaptation produces compositions. In compositional adaptation the nodes and edges from the original input spaces are broken apart into individual pieces or components. These pieces can then be strung back together to create compositions based on the given mapping, treating any mapped elements as equivalent types. For example the edge *on* from Input 2 in Figure 6.1 connects the nodes *boat* and *water*. But because of the mapping of *house* and *boat*, the compositional adaptation can attach the node *house* in the place of *boat*, leading to the composition "house on water". One can intuitively think of this process as like piecing together a jigsaw puzzle, with the edges and mapping defining how pieces (nodes) can interlock. The process of building a composition chooses nodes and edges to add to an initially empty composition based on currently open slots. This choice can be made randomly or according to some domain-specific goal or heuristic. This process can stop whenever there are no unconnected edges, meaning one can end up with smaller or larger graphs than the input spaces, which accounts for most of compositional adaptations expressive power and what differentiates it from amalgamation. However, depending on input and mapping, the output of compositional adaptation can be understood as a superset of the output of amalgamation.

6.2 Output Space Comparisons

To help clarify the differences between the three historic combinational creativity approaches I visualize the space of their potential outputs in Figure 6.2. This is an illustrative, theoretical abstraction and should not be considered to exactly define the space of possible

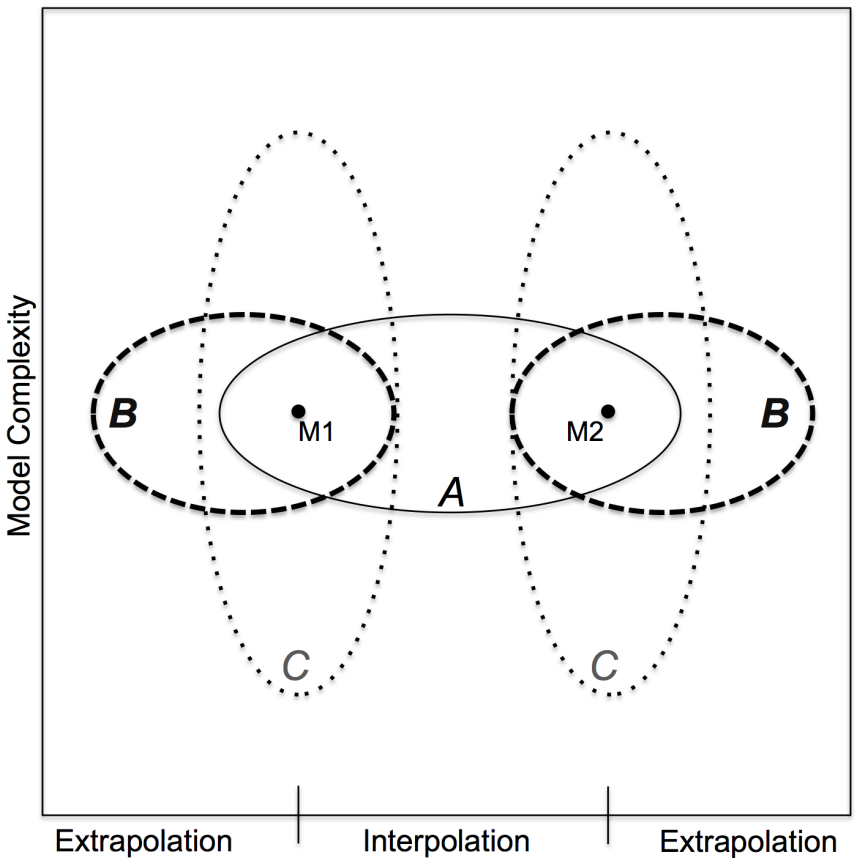

Figure 6.2: Visualization comparing the search spaces of the three historical combinational creativity approaches highlighted in this chapter.

outputs in a particular implementation of these approaches.

My focus in this dissertation is in the application of combinational creativity to machine learning in what I call invention problems. Thus, while the inputs to these combinational creativity approaches would typically be hand-authored graphs representing particular concepts or artifacts I instead focus on machine learned models. M1 and M2 thus represent an abstraction of two machine learned model inputs. The y-axis is an abstraction of model complexity, which I define as a domain-dependent measure of information present in a final model. The x-axis is based upon the inputs, with the space between them representing interpolations between the two models, different combinations of existing information present in the original models. The middle of the x-axis would then represent models that were perfect 50-50 combinations of components that already existed in the original two

models. The two ends of the x-axis then represent extrapolations off the original models, representing final combinations that include new components with new information.

The oval marked with an A in Figure 6.2 represents the space of possible amalgams, largely covering possible interpolations between the two inputs. However, it is possible to end up with amalgams that vary slightly in terms of complexity, given that high-information components might be mapped to low-information components, thus allowing for amalgams with more or less total information compared to the inputs. Similarly, it is possible to end up with some information entirely distinct from the information present in the original inputs, given input models with a large number of components that cannot be mapped. In this case, because it is necessary to include all possible components from the input in an amalgam it is possible amalgams would be output that end up with new kinds graph structure not present in the original inputs.

The two ovals with the dashed lines and B's cover the space of possible output blends. Given the extent to which blends create components not present in the original input, the two ovals are centered further along the extrapolation portion of the x-axis. However, it is possible to get more amalgam-like mappings depending on how the original generic-space mapping is converted into a blend space mapping. As with amalgams, it is also possible for the complexity of the final output to differ somewhat.

The two ovals with the dotted lines and C's cover the space of possible output compositions. The ovals are centered on the inputs, varying along the y-axis equally in either direction. This indicates that compositions primarily vary from their input by being either more or less complex, directly related to the piece-by-piece process by which compositions are built, allowing for more or less final components than their inputs. Given that compositions output tends to be a superset of amalgam output, it is also possible for compositions to interpolate between models. However, in an edge case with a large number of unmapped components, an amalgam can represent output more evenly composed of both inputs than a composition. Similarly to an amalgam, a composition can represent an extrapolation off

an input model. But in certain edge cases with a large degree of mapped components, an amalgam can represent even further extrapolations (e.g. more new information in the final output combination).

Across all three of these highlighted approaches it is possible reach output that is equivalent to either of the original two inputs. This might seem counter-intuitive, but it is as a side-effect of how these combinational creativity approaches attempt to retain helpful structure from their inputs. However, it is also not possible for any of these approaches to reach particular areas of this theoretical space, as in the corners of Figure 6.2. While one could employ multiple rounds of these approaches, such as first running conceptual blending of the input and then compositional adaptation between the output blend and one of the inputs, this is inefficient and lacks theoretical elegance. Further this does not address other fundamental limitations of these historic approaches, such as their inability to handle arbitrary numbers of input or continuous variables.

6.3 Problem Statement

The existing combinational creativity approaches have five limitations that make them ill-suited to the stated goal of this dissertation: applying combinational creativity to machine learned models. First, outside of the three highlighted in this section, all of the prior approaches are domain-dependent, which makes it difficult to accurately evaluate their utility. Second, the majority of prior approaches including those highlighted in this chapter handle only two inputs by default. There are ways of extending this, for example by combining two inputs in one step and then the output of that process with another input. However, this is a somewhat stilted way of combining input, which does not match the human inspiration of combinational creativity (e.g. people can imagine a fake animal made up of far more than two existing animals combined).

Third, all these prior approaches are severely limited in their space of possible outputs. As visualized in Figure 6.2, there is no way to reach any of the corners of this space. While

this may be ideal in cases where one wants to restrict or predict the output of a system, it limits the potential final models one can reach. Fourth, and speaking of limitations, all of these approaches are designed to output a single correct final combination. The fact that they each can create a space of output combinations has been seen as a drawback, historically. This is another limiting factor that impacts the kinds of final models it is possible to reach. Finally, all of these approaches have been designed to work with human-authored knowledge representations. Therefore, it makes sense to design a combinational creativity approach from the start under the assumption the input will be noisy, machine learned data.

6.4 Conceptual Expansion

Conceptual expansion differs from these existing approaches in a few key ways. First, it can take an arbitrary number of inputs, including only a single input as I will demonstrate in Chapter 9. Second, it clarifies the combination of individual components, instead of relying on domain specific measures, randomness, or general heuristics like including the most information possible. Instead, conceptual expansion directly defines what combination of features from each mapped component will be present and to what extent they will be present in the final expanded component. This specificity allows conceptual expansion to better handle noisy input, like machine learned data, where it not possible to assume that all of the input and its structure is valuable. Given that conceptual expansion focuses on the combination of the atomic units of its input, components in this running example, it requires either a set final global structure or the application of a compositional adaption-esque piece-by-piece construction for complete expansions. In the former case, if there is a set final graph structure one simply creates expanded components for each node and edge of this graph.

The conceptual expansion function defined for a given component is:

$$\mathrm{CE}^{\mathrm{X}}(F, A) = a_1 * f_1 + a_2 * f_2 ... a_n * f_n \tag{6.1}$$

Where $F = \{f_1 ... f_n\}$ is the set of all mapped components and $A = \{a_1, ... a_n\}$ is a filter representing what of and what amount of mapped feature f_i should be represented in the final conceptual expansion. X is the goal conceptual component that the conceptual expansion process is attempting to represent. This goal can either be a literal component, whatever that means for a particular domain, or it can be defined via domain-specific heuristics or constraints. The exact shape of a_i depends upon the feature representation. If features are symbolic, a_i can have values of either 0 or 1 (including the mapped feature or not), or vary from $-\delta$ to δ where δ is domain-specific threshold, by default 1. For example if one imagines two mapped nodes, one with a value 5.0 and one with a value -5.0, one can imagine a number of expanded nodes with all possible values between 5.0 and -5.0. If the node values were categorical values of A and B we could expand final nodes of either A, B, AB, BA, or an empty value depending on the a values and their ordering.

I use the addition symbol ($+$) not to indicate a literal summation but a combination of features within the structure of the particular component. For example if X specifies a list, each $a_i * f_i$ would be appended to the final list, but if X specified a matrix then this would literal matrix addition. Thus $+$ in this case indicates the data representation-appropriate addition function.

I visualize an example of this function applied to a set of black and white pixel face components in Figure 6.3. In this figure there are three mapped components, represented as matrices of black and white pixels. Each pixel then is an individual feature f_i of one of these mapped components f, with the set of all three faces being F. The three matrices to the left of the three pixel faces represent A the set of a values that tell us what of each mapped component f to make use of in the final expanded component (the duck face on

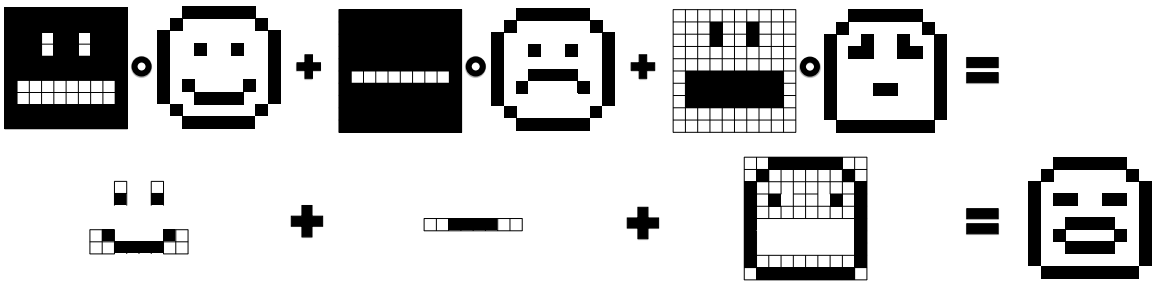

Figure 6.3: Visualization of the conceptual expansion function on a matrix.

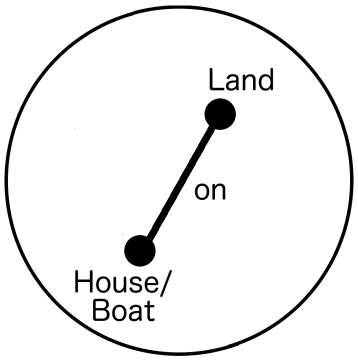

Figure 6.4: Example of conceptual expansion output given the input and mapping used as an example throughout this chapter.

the far right).

One can understand conceptual expansion in the case with all symbolic values and without a fixed final structure as a hybridization of conceptual blending and compositional adaptation. In particular conceptual expansion creates the individual blended elements based on the given mappings as in conceptual blending, then constructs expansions piece-by-piece according to these mappings as in compositional adaptation. One can understand conceptual expansion's output in this case as representing a superset of output of both of these techniques, which further includes output neither technique could generate. For example consider the final graph in 6.4. It contains a "blended" node in the inclusion of House/Boat, which could not be output via compositional adaptation. But is also smaller in size than its inputs, which could not be output via conceptual blending. However, conceptual expansion primary strength, the ability to determine how a combined component is constructed via f

and a values is not present in a purely symbolic

6.5 Case Study: House Boat

Thus far I have given a single example output for each approach from the House and Boat inputs, however each combinational creativity approach is capable of creating many more outputs. In this case study I present analysis of the entire output space of each of these approaches given these two hand-authored inputs [8]. This is meant as a more thorough comparison of the potential outputs of these approaches. Further, it stands as a point of contrast to the application of machine-learned data as input, which I will demonstrate in a case study later in this chapter. In particular it demonstrates the differences between human-authored and machine-learned input and why one might choose a different combinational creativity approach for a particular kind of input.

I created a random baseline for comparison purposes by creating a random graph and assigning random values from the input elements and all possible blends of the input elements to the edges and nodes of each graph. The output of this approach can be understood as a superset of all possible output of the other approaches, but without any of the benefits that might come from leveraging the existing structure of the input graphs. For example one output for the House and Boat inputs would be "Water on Water". I limited the size of the possible output graphs to the maximum size of the two input graphs. Otherwise unbounded the random baseline output space would be infinite. While an individual combination from this random baseline is constructed randomly, because I am generating all possible outputs it can be understood as equivalent to the space of all possible symbolic combinations over its input. For both conceptual expansion and this random baseline I limited the maximum number of components that could be combined for a single combined component to two, in order to better compare their output to existing combinational creativity approaches.

6.5.1 Metrics

The size of the output spaces of these approaches makes directly reporting each potential output ill-advised. Instead I employ a number of summarizing metrics to compare the output. As stated in the introduction Boden defines creativity in terms of surprise, novelty, and value [3]. Given the eventual goals of this document I employ measures of novelty and value as my two summarizing metrics. I do not attempt to computationally represent a notion of surprise in this study, given a lack of clarity as to what surprise would mean in this context without human judgement. Given the number of comparisons required directly involving human participants to make these comparisons was not possible.

To represent novelty I measured a normalized edit distance from the two input graphs. I first determined the edit distance between both inputs and the output graph. I took the smallest of these values and then divided it by the size of the larger of the two input graphs. In this case study both inputs have the same size, with three nodes and two edges for a total of five components.

I measured value in terms of a final output that included these three elements given that the inputs describe human entities that interact with objects in some context. I decided that the most important part of the output was the inclusion of an object ("House", "Boat", or "House/Boat"), given the way this example was originally described by Goguen and Fox [8]. Thus the inclusion of an object in the output graph increased the value by 0.5, and the inclusion of the other two elements increased the value by 0.25. The value metric thus had a maximum score of 1.0.

Consider the conceptual blending output presented already: "Resident/Passenger rides/lives in a House/Boat on the Water". This output would receive a 1.0 for value given that it has all the required elements. All but two of this output's components ("on" and "Water") are novel to the input as they are blended components. The smallest edit distance is therefore from the input House graph, with a value of three. Thus the final novelty metric is three divided by five or 0.6.

To better clarify the output of these approaches I chose to bucket the output into four categories based on high and low values of novelty and value. I employ 0.5 as a cutoff, with all values equal or less than this for either metric considered low and all values above this considered high. Thus this lead to a total of four possible categories. I refer to the high value and high novelty category as high quality output. While these metrics are not exhaustive in terms of determining high quality output, any high quality output must also have high values of these metrics.

Ultimately in comparing these approaches what matters is two particular qualities of their space of possible outputs. First, the output space of an ideal approach should be biased towards high quality output. This way whatever approach is used to select individual output from the space, whether that be a random sampling, rules-based, or optimization-based approach, will be more likely to find high quality output. This first quality can be considered equivalent to the concept of precision, if we consider high-quality output as relevant instances and all the output of the approach as retrieved instances. Second, The second quality of the output space of any particular approach is that it has good coverage of the space of possible high quality outputs. This quality is not measurable given an infinite output space, but can be measured given certain constraints on the output space. One can measure this quality in this case study given that the two inputs are small and symbolic. This second quality can be considered equivalent to the concept of recall, where we compare the number of relevant instances (high quality output) of each particular approach to the total number of relevant instances (high quality output of the random baseline).

6.5.2 Case Study Results

I summarize the output of all the approaches in terms of the four categories in Table 6.1. Low V indicates low value and Low N indicates low novelty. Thus, amalgamation output 16 high value and low novelty graphs.

There are a few clear takeaways from these initial results. The first is that the random

Table 6.1: The total number of output for each category for the House Boat case study.

Approach	Low V/Low N	Low V/High N	High V/Low N	High V/High N
Amalgams	0	0	**16**	0
Blends	0	0	22	**29**
Compositions	0	**18**	16	0
Expansions	0	45	**48**	33
Random	94	182	544	**2,299**

Table 6.2: Precision and recall for each approach for the House Boat case study.

	Amalgams	Blends	Compositions	Expansions	Random
Precision	0	0.57	0	0.26	0.74
Recall	0	0.01	0	0.01	1

baseline far outperforms any of the other approaches, creating nearly two orders of magnitude more high quality output. There are two likely reasons for this. First, the quality and simplicity of the hand-authored input graphs. The input graphs already focus the concepts down to the important pieces, meaning any arbitrary combination is likely to be high quality. Second, my metric for valid is under-specified, in particular it does not account for semantic issues that a more sophisticated evaluator might consider. For example, the output "Land/Passenger in a Land/Water lives in Land/House" would be a highly novel, highly valuable output, but it is semantically challenging to say the least. Conceptual expansion does create more high quality output than any of the historic approaches and would not have the same semantic issues due to the mapping, but its output is slightly biased towards high value and low novelty output.

Table 6.2 contains the precision and recall values for each approach. The precision results suggest a clear preference across the existing combinational creativity approaches for conceptual blending in cases with hand-authored input like this example. This is to be expected given its historic popularity for problems dealing with the combination of hand-authored input [44, 8, 69]. However, the random baseline has the highest precision value. This suggests that the space of all possible output has a greater percentage of high quality outputs compared to any other approach given human authored-inputs like the House and

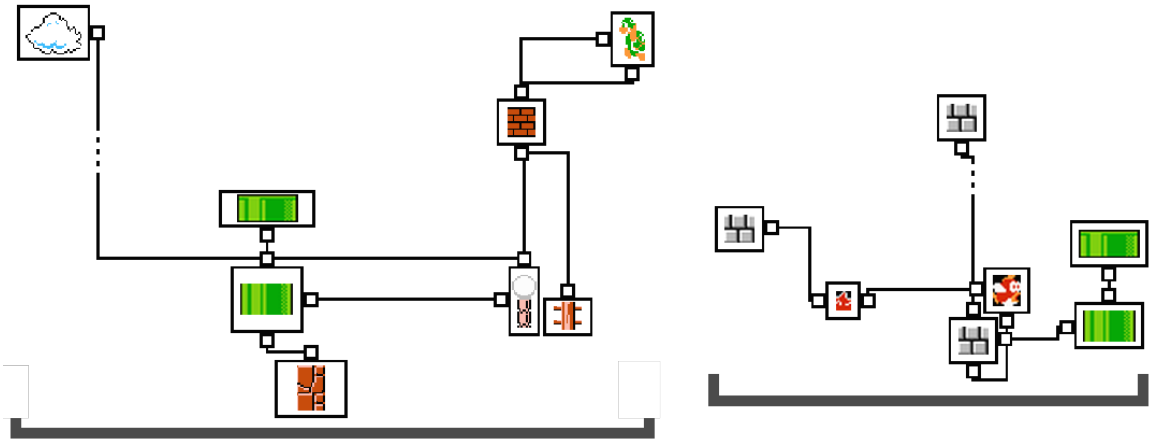

Figure 6.5: Example of the machine-learned level graphs for overworld (left) and castle (right) type levels.

Boat graphs. The random baseline has perfect recall given that we use its high quality output as the set of all relevant instances for calculating recall. All other approaches cover only 1% or less of this possible output. This suggests that in the case of well-authored human input the structure implicit in the input is sufficient to get high quality output using any strategy for combination.

6.6 Case Study: Combining Machine-learned Game Level Models

For the remainder of this chapter I demonstrate the impact of applying these combinational creativity approaches to machine-learned level design models. This is a much more challenging combination task than combining the well-designed human-authored input. This allows me to demonstrate the strengths and weaknesses of the approaches in a more grounded way, and clarify the differences between conceptual expansion and the highlighted, historic approaches. Further it highlights some of the differences between machine learned and human-authored input.

I collected on five of the learned S-structure graphs for this case study representing each class of Super Mario Bros. level: overworld, underground, athletic (sometimes called treetop levels), castle, and underwater. I visualize the overworld and castle S-structure

graphs in Figure 6.5. The level design model learning process learns many L nodes for each of these level types. I selected the single simplest L node of each type from which to derive the S-structure graphs for this case study. I chose the simplest L node for the benefit of the three highlighted combinational creativity approaches, given that they are designed for simpler, hand-authored input. The S-structure graphs for the chosen L nodes had an average size of six nodes and seven edges, which makes this problem as close as possible in complexity to the House Boat case study.

Applying conceptual expansion to this therefore means recombining the individual S nodes, so each S node is an f value. Each S node of a conceptual expansion output is then a combination of the S nodes from the input graphs, dependent on their a values. However, I limit the number of f values that can be combined to advantage the historic combinational creativity approaches.

I use the same mapping for all of the combinational creativity approaches, the hierarchical information adapted from the classes used by Summerville and Mateas [74] (e.g. "enemy" as a parent of all game enemies, "solid" as a parent of all unbreakable level components, etc.). In this way all enemy nodes are mapped together, all solid nodes mapped together and so forth. This type of hand-authored, hierarchical knowledge base is more traditionally applied to combinational creativity problems. I chose to employ it here for the benefit of the three historical approaches. I made use of every pair of input graphs, making for a total of ten combination tasks as opposed to the single combination task in the prior case study. I generated all possible unique output combinations for each approach. This can be understood as analogous to exhaustive procedural content generation [167]. In the case of combined S nodes I combined both sets of G and D nodes and recalculated the relevant probability distributions.

I only made use of inputs in pairs and again limited the random and conceptual expansion approaches to only combine at most two input components for each final output component. This benefits the historic approaches, but also constrains the random output for

Table 6.3: Average and standard deviation of the number of output of each category. I have marked in bold the largest average output category for each approach.

Approach	Low V/Low N	Low V/High N	High V/Low N	High V/High N
Amalgams	1.4±2.5	2.1±4.3	**6.5±3.8**	2.7±6.5
Blends	22.4±27.9	**48.0±84.3**	36.9±33.0	26.7±26.9
Compositions	2.2±2.2	39.5±17.5	11.2±11.5	**91.4±35.8**
Expansions	46.6±121.3	**200.8±242.9**	65.4±151.0	154.4±145.2
Random	108.8±187.5	**1117.6±2231.8**	130.5±196.5	545.8±663.0

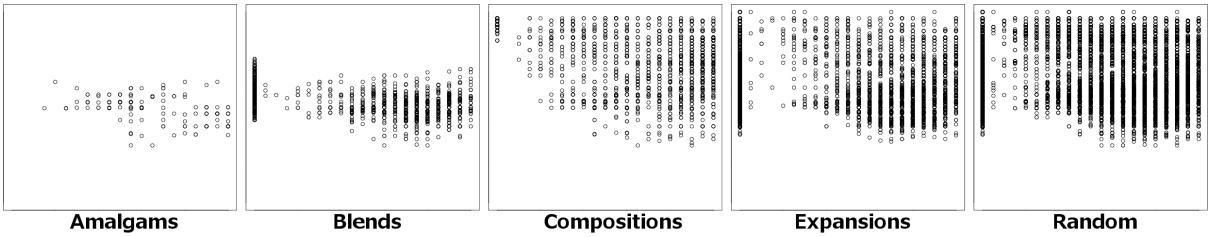

Figure 6.6: Scatterplots of the output values for each approach, the x-axis is value (playability) [0,1] and the y-axis is novelty [0,1].

the purposes of calculating recall. I anticipate the results demonstrated in this case study would only be more extreme given varying input set sizes and without this constraint.

6.6.1 Results

Table 6.3 contains the average output and standard deviation across the ten pairs of inputs in this case study. In terms of raw numbers conceptual expansion had a significantly higher distribution of outputs compared to all the historical approaches according to the Wilcoxon Mann-Whitney U test ($p < 0.01$), but the random baseline produced significantly more output than it.

For a more detailed representation of the output space I followed the practice of expressive range [168]. I present scatterplots of the generators output in Figure 6.6 with value (playability) on the x-axis and novelty on the y-axis. These values reflect the intuitions expressed above for the approaches. For example: output amalgams have generally high value but low novelty, blends have consistent bias towards high value, and compositions have a bias towards high value and high novelty. Expansions are more even across all four

Table 6.4: Average and standard deviation for precision and recall for each approach.

	Amalgams	Blends	Compositions	Expansions	Random
Precision	0.11±0.17	0.28±0.31	0.63±0.63	0.40±0.42	0.36±0.39
Recall	0.02±0.05	0.10±0.13	0.42±0.40	0.54±0.52	1.00±0.00

categories, with a bias towards higher novelty. The random output has a strong bias towards non-playable, highly novel output, which matches my expectation given it doesn't make use of any of the structural information from the inputs. While the expansions and random output may seem the same at first glance, the expansions output space is more similar to if one overlapped the blends and compositions. In particular, not less density of points towards the top, particularly towards the top left (high novelty, low playability).

In the last case study I employed two summarizing metrics, based two of the three elements of creativity identified by Boden [17]. In this case study I employ the same novelty metric, but employ a new value metric given the case study domain of level design.

Given that the inputs are generative level design models I measured value in terms of the playability of the generated level chunks of the output combinations. I approximated the playability of the generated level chunks by generating a hundred level chunks for each output combination and determining the percentage that could be completed with an A* Mario agent (inspired by the A* agent from the Mario AI competition [165]). This agent represents perfect play and therefore is only an approximation of the playability of the output level chunk. However, it is sufficient for the purposes of this study given its scale.

I include the average precision and recall across all ten input pairs for each approach in Table 6.4. Compared to the prior case study, random and conceptual blending score less well in terms of precision. This indicates that their output spaces have much sparser high quality output. I take this as a primary difference between well-designed human input and messy, machine-learned input. In comparison when it comes to precision, compositional adaptation does much better. This follows from the fact that the S-Structure Graph represents a focused version of an L node that only retains the most important edges and

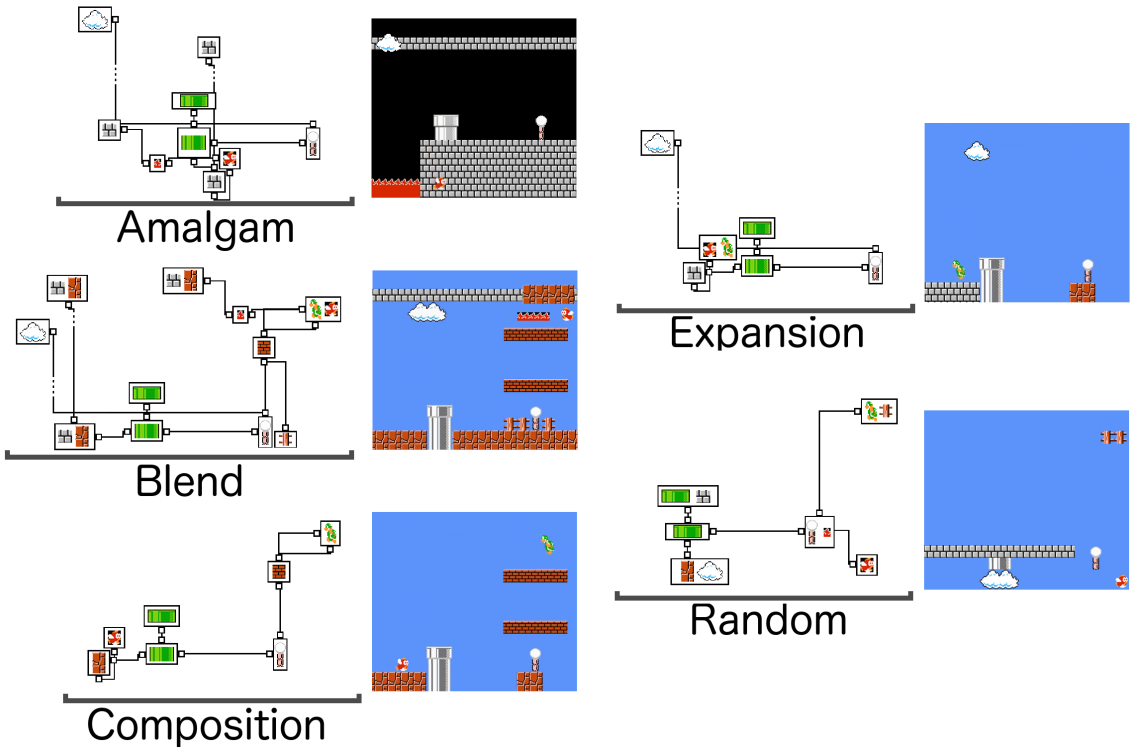

Figure 6.7: Random selection of a high novelty, high playability output graph from each approach for the overworld-castle combination, with an associated generated output level chunk.

nodes. Given that compositional adaptation focuses on reusing existing relationships to create novel combinations, it is well-suited to this domain. Conceptual expansion also does much better in this domain, outperforming all other approaches in terms of this measure outside compositional adaptation.

Again, I treat the high quality of the random baseline as all relevant instances and thus it achieves perfect recall. However, conceptual expansion has the best average recall of any of the non-random baseline approaches. This indicates it has the best coverage of high quality output of any of the non-random baseline approaches. Conceptual expansion thus can generate the majority of high quality output possible, while being more likely overall to generate high quality output than a purely random baseline.

In order to give a more clear understanding of the performance of these different approaches I present a random example of a high quality output for each of the combinational creativity techniques for the overworld+castle input pair. I also include a single generated

output level chunk for each output graph, though I note each graph is capable of generating a wide variety of unique level chunks. As one can see in Figure 6.7 each graph is distinct. The amalgam and blend are the most complex, due to the requirement that they make use of the maximum amount of elements possible. Of the remaining three, the composition and expansion differ in that the expansion has a blended element with the two "enemies" combined. Lastly, while the random output is technically novel and playable it involves merging clouds with the ground and having floating fences, which leads to generated levels that differ significantly from the standard Super Mario Bros. levels. As stated previously these metrics do not reflect the entirety of S-Structure graph quality, and in fact much of the output of the random baseline looks aesthetically noisy in this same way.

6.7 Discussion

The initial results from these case studies identify cases in which each presented combinational creative technique could be applied. Amalgamation presents a safe option when one lacks the ability to differentiate between valuable and non-valuable output, but only produces a small number of additional unique models. Conceptual blending produces high quality output in cases with well-designed, human-authored input. Compositional adaptation produces high quality output in cases where the input encodes only the most important information and minimal output is acceptable. Conceptual expansion seems to handle machine-learned data better than these other approaches given that it has better coverage of the high quality output space (better average recall than the non-random approaches) while still being more likely to find high quality output than a random baseline (better precision than the random baseline). One can expect this disparity to only become greater without the limitations placed on conceptual expansion described above.

There are domains that are more or less suited to the notion of blended elements. For example, when the input variables have numeric values (e.g. does 3 and 4 make 7, 34, 43, or something entirely different). This is a major element of statistical machine learning

models. Conceptual expansion produced far more unique output graphs compared to the historic approaches, and could therefore prove helpful in cases where one can automatically differentiate between low and high quality content and desire a large possible space of outputs that still leverages the important structure of its inputs. Further I again note that conceptual expansion naturally can handle non-numeric input and arbitrary sets of inputs, which none of the other historic approaches can handle in a standard application.

I note there are limitations with this case study, particularly with my choice of playability as a measure of level design model value. Even if only twenty percent of the output from a level design model is playable, if one can differentiate between playable and unplayable content then the unplayable content can be avoided. Further, playability does not account for many important aspects of a level, such as style and subjective player experience. One can see this most clearly in the results of the random baseline, which produced many levels that were technically playable but from my perspective in no way resembled Super Mario Bros. levels.

6.8 Conclusions

In this chapter I initially define conceptual expansion and demonstrate it in comparison to existing combinational creativity approaches in two case studies. Conceptual expansion allows one to address problems with combinational creativity that include an arbitrary amount of input, continuous variables, and input content that one does not wish to replicate in the combined output. These features are especially important when combining machine learned models. In the case studies in this chapter I intentionally handicap conceptual expansion. Despite this, the approach outperforms the existing approaches in terms of the amount of high-quality output it can produce given machine-learned input. While a random combination produced an even larger number of high-quality output conceptual expansion had a higher precision in delivering high quality output. This indicates that sampling from the space of possible conceptual expansions would more likely lead to high quality output

compared to this random baseline, demonstrating the importance of leveraging structural information from the input.

Despite being handicapped with relatively simple input conceptual expansion still produced more output than existing combinational creativity approaches. However, despite the bias towards higher quality output, a random sampling approach is unlikely to produce consistent results. Therefore in the next chapter I introduce a search-based approach to find high-quality expansions using domain-specific heuristics. This also allows conceptual expansion to automatically determine the correct combination of input to employ for creating a novel output component.

This chapter represented a thorough investigation into the video game level invention problem. In the next chapter I employ conceptual expansion to variations of the full video game invention problem. Towards this I employ a novel, graphical representation that combines both the level design models and rules learned from gameplay video. I leverage a search-based approach in order to find high quality output from the space of possible conceptual expansions of games in this representation.

CHAPTER 7

THE VIDEO GAME INVENTION PROBLEM

Automated game design, generating entire games from scratch, has remained a key challenge within the field of Game AI. The video game invention problem—the problem of creating novel, surprising, and valuable games—can be understood as analogous to this challenge. Game design and development requires a large amount of expert knowledge, in terms of design and coding skills. This skill requirement serves as a barrier that restricts those who might most benefit from the ability to make games. Researchers have touted automated game design as a potential solution to this issue, in which computational systems build games without major human intervention. The promise of automated game design could not only democratize game design, but allow for educational, scientific, and entertainment applications of games currently infeasible given the resource requirements of modern game development. However, up to this point automated game design has relied upon encoding human design knowledge in terms of authoring parameterized game design spaces or entire games for a system to remix. This authoring work requires expert knowledge, and is time intensive to debug, which limits the applications of these automated game design systems.

In this chapter, I explore conceptual expansion applied to automated game design in three distinct variations of the video game invention problem. In the last chapters I combined only level design knowledge from within a particular game to create new level types for that game. In this chapter I combine both knowledge about level design and game rules for different games to create entirely new games. Prior automated game design systems have relied on hand-authored or crowd-sourced knowledge, which limits the scope and applications of such systems. Instead I draw on the machine learned knowledge of level design models (Chapter 3) and mechanics (Chapter 4), as elements of a new representation

I call a *game graph*. I then apply conceptual expansion to this game graph representation, based on a knowledge base of three existing game graphs.

This chapter is organized as follows. First, I cover related, prior attempts at automated game design relevant to this chapter. I then briefly review both the learned level design and rule learning processes for the purpose of informing the game graph representation. After introducing this representation I cover three distinct variations involving conceptual expansion and game graphs. For all three variations I make use of a new process for finding a single output from the space of possible conceptual expansions, which optimizes the final conceptual expansion according to a goal or heuristic. I call this process *conceptual expansion search*.

The first variation recombines two of the existing game graphs to recreate an existing third game graph, based on a partial-specification of that game graph. This can be thought of as a p-creative version of the video game invention problem. I evaluate this via a comparison to conceptual blending, genetic algorithm, and K-nearest neighbors baselines. Notably, this first variation only recreates game graphs and does not convert these game graphs into a piece of software playable by a human.

The second variation recombines the three existing game graphs to create novel games—instead of recreating a held out game graph—via optimizing a simple heuristic. I convert the output game graphs into playable games in a human-in-the-loop framework. This human-in-the-loop framework is necessary as the system has no understanding of game visuals outside of existing sprites, thus it only represents entities of a new game as differently sized rectangles. While this process creates high-quality games, it obfuscates whether the value of the system comes from the artificial or human part.

The third variation removes the human from the loop. Instead visuals in the game graph are handled by having the entire process target a new spritesheet without an associated game. This spritesheet was made open source to allow anyone to make a game with it, but does not have an associated game or set of mechanics. Thus, many possible games can be

made with this spritesheet. This represents the most clear attempt to solve the video game invention problem. I evaluate the system in a human subject study, comparing the games created by the conceptual expansion approach to games created by humans and by the three existing, highlighted combinational creativity approaches (amalgamation, conceptual blending, and compositional adaptation). All the games in this study were created using this same, target spritesheet.

7.1 Background

In this section I briefly review prior work relevant to the attempts at video game invention presented in this chapter. Automated game design has been an active area of research for over the past twenty years, with the earliest work investigating generation of chess-like games [84], card games [169], board games [85] and simple arcade games [86]. Recent work has focused on researchers defining a parameterized space of possible games either explicitly or through authoring a game representation, which can then searched or constrained [170, 171]. This common approach represents a heavy authoring burden along with requiring expert knowledge to design a good representation or search space. Cook's ANGELINA artificial game designer is perhaps the most well-known automated game design system [172, 91]. While there have been many versions, they all rely on evolutionary search to optimize particular types of games. Relevant to this work, Gow and Corneli [68] proposed applying conceptual blending to recombine existing games in the Video Game Description Language (VGDL) [94], but as of this writing no system has been built.

The major difference among the three variations of conceptual expansion game generation in this chapter is the selection criteria. In the first variation I employ a partially specified goal, with the distance to this goal as the selection criteria for what expansion from the output space to return. In the second I employ a simple heuristic and human judgement, and in the third I employ a more complex heuristic. Thus this work touches on the problem of content selection, how to choose the best or most creative content from

a set of options. This problem has been studied in computational creativity across many domains including narrative [173], music [174], and poetry [175]. However, due to the lack of a clear dominant content selection strategy in games I employ heuristics inspired by the metrics employed in Chapter 6. I discuss the implications of the choice of heuristic further in this chapter.

7.2 Game Graph Representation

For the video game invention problem I employ a novel representation I call a game graph, which encodes both the learned level-design information and the learned game ruleset information. In this section I walk through the entire process of building these game graphs. This representation is partially inspired by the graph-based concept representation of [10].

The full process is as follows: I take as input gameplay video and a spritesheet. A spritesheet is a collection of all of the images or sprites in a game, including all background art, animation frames, and components of level structure. I run image processing on the video with the spritesheet to determine where and what sprites occur in each frame. Then, I learn a model of level design and a ruleset for the game. I then merge the representations of level design and game ruleset into what I call a *game graph*. A game graph is a graphical representation in which each sprite is treated as a node with edges representing the level design and rules of the game. Finally, I apply my conceptual expansion approach on these game graphs to produce a new graph which can be re-represented as a playable game. In the following subsections I review the pertinent aspects of the techniques I use to learn models of level design and game rulesets, and how I create game graphs from the output of these techniques.

7.2.1 Level Design Learning

I use the technique described in Chapter 3 to learn a generative model of level design. At a high level this technique derives a hierarchical graphical model or Bayesian network that

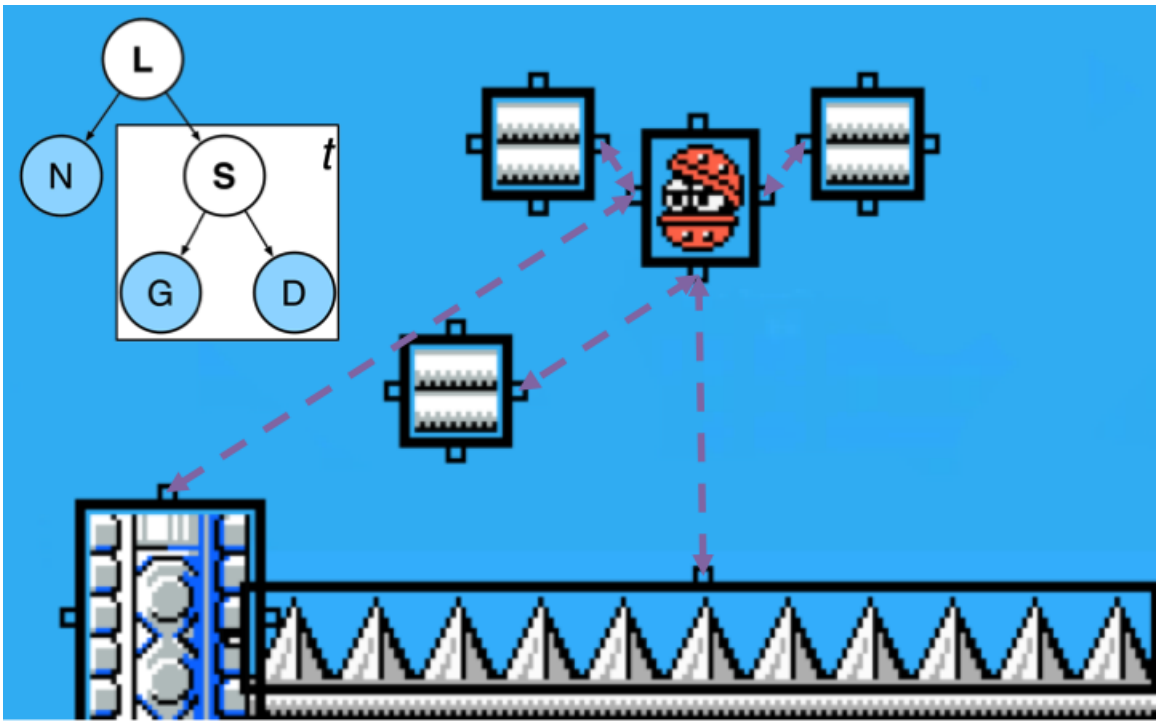

Figure 7.1: A visualization of the model (left) and basic building blocks from a subsection of the NES game Mega Man.

represents probabilities over level structure from gameplay video. I visualize the abstract model and two base components in Figure 7.1 for the game Megaman. There are a total of five types of nodes in the learned network:

- G**:** Distribution over geometric shapes of sprite type t. In Figure 7.1 each box contains a G node value.
- D**:** Distribution over relative positions of sprite type t. In Figure 7.1 all the purple lines represent a D node value.
- N**:** Distribution of numbers of sprite types in a particular level chunk. For example in Figure 7.1 there are eleven spikes, three bars, one flying shell, etc.
- S**:** The first hidden value, on which G and D nodes depend. S the distribution of sprite styles for sprite type t, in terms of the distribution of geometric shapes and relative positions. That is, categories of sprites. For example, in Figure 7.1 there are

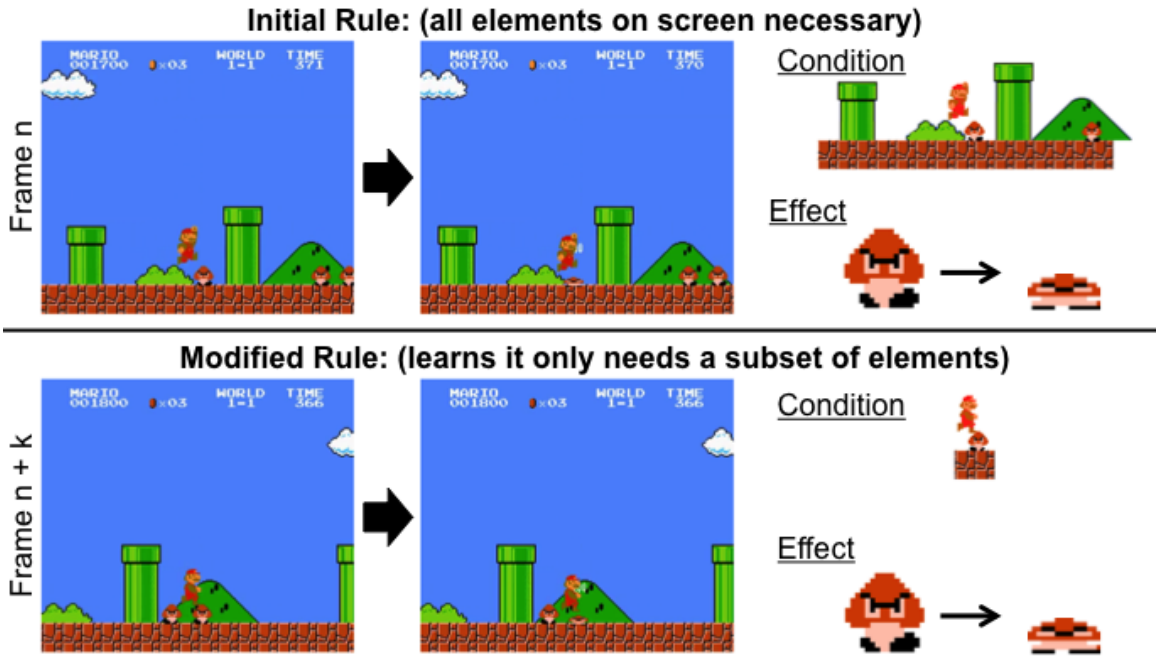

Figure 7.2: A visualization of two pairs of frames and an associated engine modification.

three bars, but they all have the same S node value.

- L**:** Distribution over level chunks.

L nodes are learned by a process of parsing gameplay video and iterative, hierarchical clustering. A probabilistic ordering of L nodes is learned to generate full levels.

7.2.2 Ruleset Learning

I employ the approach as described in Chapter 4 to learn rules from gameplay video. This technique takes as input gameplay video and represents each frame as a list of conditional facts that are true in that frame. Whereas content modeling assumes the game is already known, this approach does not make this assumption. The fact types are as follows:

- $Animation$**:** Each animation fact tracks a particular sprite seen in a frame by its name, width, and height.

- $Spatial$**:** Spatial facts track spatial information, the x and y locations of sprites on the screen.

- $RelationshipX/RelationshipY$**:** The RelationshipX and RelationshipY facts track the relative positions of sprites to one another in their respective dimensions.
- $VelocityX/VelocityY$**:** The VelocityX and VelocityY facts track the velocity of entities in their respective dimensions.
- $CameraX$**:** Tracks the camera's x position.
- $CameraY$**:** Tracks the position of the camera in the y dimension.

The algorithm iterates through pairs of frames, using its current (initially empty) ruleset to attempt to predict the next frame. When a prediction fails it begins a process of iteratively updating the ruleset by adding, removing, and modifying rules to minimize the distance between the predicted and actual next frame. The rules are constructed with conditions and effects, where conditions are a set of facts that must exist in one frame for the rule to fire and effects are a pair of facts where the second fact replaces the first when the rule fires. I visualize two examples of this process in Figure 7.2 for the game Super Mario Bros.. In the top row the difference of the goomba being squished or not is accounted for with a new goomba squishing rule that adds as a condition all facts from the first frame. Later when another pair of frames is encountered, that initial rule is modified to be more general. This occurs as the search heuristic prefers smaller, less complex rulesets. The end result of this process is a sequence of rules that allows one to forward simulate the entire game state.

7.2.3 Game Graph

The output of the level design model learning process is a probabilistic graphical model. The output of the ruleset learning process is a sequence of formal-logic rules. I combine both of these into a single representation I call a *game graph*. This game graph represents a minimal representation of an entire game.

The construction of the game graph is straightforward. Each sprite in a spritesheet for a particular existing game becomes a node in an initially unconnected graph. Then all the

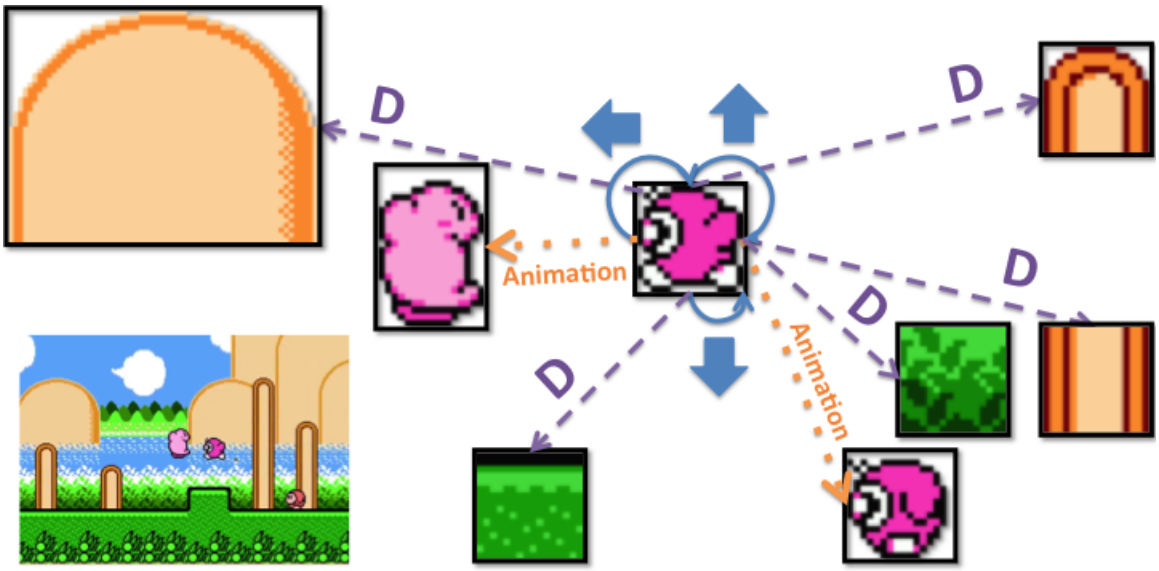

Figure 7.3: A subset of the game graph for one Waddle Doo enemy sprite from Kirby's Adventure and a relevant gameplay frame.

information from the level design model and ruleset representations is added as edges on this graph. There are five distinct types of edges that contain particular types of values:

- **G:** Stores the value of a G node from the level design model, represented as the x and y positions of the shape of sprites, the shape of sprites (represented as a matrix), and a unique identifier for the S and L node that this G node value depends on. This edge is cyclic, pointing to the same component it originated from. I note one might instead store this information as a value on the node itself, but treating it as an edge allows us to compare this and other cyclic edges to edges pointing between nodes.

- **D:** Stores the value of a D node connection, represented as a vector with the relative position, the probability, and a unique identifier for the S and L node that this D node value depends on. This edge points to the equivalent node for the sprite this D node connection connected to.

- **N:** Stores the value of an N node, which is a value representing a particular number of this component that can co-exist in the same level chunk and a unique Identifier for its L node. This edge is cyclic.

- **Rule condition:** Stores the value of a particular fact in a particular rule condition, which includes the information discussed for the relevant fact type and a unique identifier for the rule it is part of. This edge can be cyclic or can point to another component (as with Relationship facts).

- **Rule effect:** Stores the value of a particular rule effect, which includes the information for both the pre and post facts and a unique identifier for the rule it is part of. This edge can be cyclic or can point to another component.

I visualize a small subsection of a final game graph for the Waddle Doo enemy from Kirby's Adventure in Figure 7.3. The cyclic arrows in blue with arrows represent rule effects that impact velocity (I do not include the full rule for visibility). The orange dotted arrows represent rule effects that impact animation state. The dashed purple arrows represent D node connection edges. This is a very large graph, with dozens of nodes and more than a hundred thousand edges. It contains all information from the learned level design model and game rulesets. That is, a game can be reconstructed from this graph. Further, the graph structure can be manipulated in order to create new games by adding, deleting, or altering the values on edges.

The only missing piece of information in this game graph is the learned level transition graph needed to string sequences of generated level chunks together. However, this only constrains the level generation process towards certain sequences of level chunks and is not strictly required to generate levels. Thus because of the nature of the first evaluation which strictly looks at game graphs and not playable output games, it is exempt from this implementation. This was largely done due to the baselines of the first evaluation and because of the increase in the size of the game graphs its inclusion would cause. For the second variation I discuss how it is integrated into the game graph structure when it is required to output human-playable games.

7.3 Variation 1: Goal-Driven Conceptual Expansion over Game Graphs

The size, complexity, and lack of uniformity of the game graph representation makes them ill-suited to generation approaches like statistical machine learning or rule-based systems. Instead I apply conceptual expansion to these game graphs. As I reminder the conceptual expansion function is:

$$\mathrm{CE}^{\mathrm{X}}(F, A) = a_1 * f_1 + a_2 * f_2 ... a_n * f_n \tag{7.1}$$

Where $F = \{f_1 ... f_n\}$ is the set of all mapped features and $A = \{a_1, ... a_n\}$ is a filter representing what of and what amount of mapped feature f_i should be represented in the final conceptual expansion. X is the concept that we are attempting to represent with the conceptual expansion. In this section the final CE^X value is some novel game graph node that is being created via the combination of existing game graph nodes F with the A values informing the formula above on what to take from each mapped existing game graph node f.

As an example, imagine that we do not have Goombas (the Mario enemy in Figure 7.2 as part of an existing knowledge base. Imagine we are trying to recreate a Goomba node with conceptual expansion ($CE^{\texttt{Goomba}}$) and we have the Waddle Doo node as seen in Figure 7.3 mapped to this Goomba node. In this case there may be some edges from this node we want to take for the Goomba node such as the velocity rule effect to move left and fall, but not the ability to jump. In this case one can encode this with an $a_{waddledoo}$ that filters out jump (e.g. $a_{waddledoo}$ = [1,0...]). One can alter $a_{waddledoo}$ to further specify one wants the Goomba to move half as fast as the Waddle Doo. One might imagine other f and corresponding a values to incorporate other information from other game graph nodes for a final $CE^{Goomba}(F, A)$ that reasonably approximates a true Goomba node.

At a high level in this process, conceptual expansion creates a parameterized search space from an arbitrary number of input game graphs. One can think of each aspect of a

Algorithm 3: Goal-Driven Conceptual Expansion Search

input : a partially-specified goal graph $goalGraph$, and a mapping m
output: The maximum expansion found according to the heuristic

```
1  maxE ← ExpansionFromGoal(goalGraph)+m;
2  maxScore ← 0;
3  improving ← 0;
4  while improving < 10 do
5      n ← maxE.GetNeighbor();
6      s ← Heuristic(n, goalGraph);
7      oldMax ← maxScore;
8      maxScore, maxE ← max([maxScore, maxE], [s, n]);
9      if oldMax < maxScore then
10         improving ←improving +1;
11     else
12         improving ←0;
13 return maxE;
```

game level design or ruleset design as a dimension in a high-dimensional space. Changing the value of a single dimension results in a new game design different from the original in some small way. Input game graphs provides a learned schematic for what dimensions exist and how they relate to each other. With two or more game graphs, one can describe new games as interpolations between existing games, extrapolations along different dimensions, or alterations in complexity. One can then search this space to optimize for a particular goal or heuristic, creating new games. From these new game graphs one can reverse engineer level design models and rulesets to create new, playable games.

This first variation of conceptual expansion over game graphs has two steps. First, a mapping is determined, which gives us the initial a and f values for each expanded node. This is accomplished by determining the best n nodes in the knowledge base according to some distance function. I make use $n = 10$ as an arbitrary starting point. For the second step I make use of a greedy hill-climbing approach I call *goal-driven conceptual expansion search*.

I include the pseudocode for the goal-driven conceptual expansion search algorithm in Algorithm 3. The input to this process is a partially-specified goal graph ($goalGraph$) which defines the basic structure of the initial conceptual expansion (e.g. the number of

nodes and some subset of the edges on those nodes) and a mapping (m) derived as I discuss above. On line one the function "ExpansionFromGoal" creates an initial set of nodes from the $goalGraph$, and for each expanded node fills in all its a and f values according to a normalized mapping in which the best mapped component has a values of 1.0, and the a values of the rest follow based upon their relative distance. I search for ten neighboring conceptual expansions and either select the best neighbor according to some heuristic $Heuristic$ or return the current expansion if it has the best value in terms of $Heuristic$.

For this first variation I make use of a heuristic of the distance from $goalGraph$ to the current game graph. I calculate the average of the distance from each component of $goalGraph$ to the closest component in the current graph, normalized between [0,1]. For component to component distance I match the closest edges of the two components, where if two edges are of different types they will have a distance of 1.0, where otherwise the distance will vary from [0,1] depending on the number of matching values. This heuristic is asymmetrical. This allows the system to measure fully defined game graphs in terms of partially defined game graphs (like the $goalGraph$). This distance function can be understood as an asymmetric Chamfer distance between game graphs.

To find a neighbor of a current conceptual expansion game graph $maxE$ in the $GetNeighbor$ function the system takes one of the following actions at random: (1) add a node to a particular component as an additional f value with random a values, (2) remove a random f and a pair, (3) for a particular expanded node change all values of a by a random value [-2,2], and (4) alter a random value of a random a value by multiplying it with a number uniformly chosen between [-2,2]. I chose [-2,2] as it the smallest whole number interval in which, with iterative alterations, we can get from any one value to any other value. Thus this process randomly samples ten possible neighbors of the current conceptual expansion game graph, and if none are better, the process stops and returns the current conceptual expansion game graph.

Table 7.1: Errors for given level design evaluation. The systems were given a level design and tasked with creating a game with a matching level design and appropriate rules.

	Mario		Kirby		Mega Man	
baseline	design err (train)	rules err (test)	design err (train)	rules err (test)	design err (train)	rules err (test)
Blend	0.394	0.404	0.609	0.130	0.422	0.413
KNN	0.285	0.180	0.645	0.270	0.456	**0.148**
GA	0.285	0.180	0.645	0.134	0.454	0.163
Expansion	**0.282**	**0.110**	**0.531**	**0.126**	**0.396**	0.189

Table 7.2: Errors for the given rules evaluation. The systems were given a ruleset and tasked with creating a game with a matching ruleset and appropriate rules.

	Mario		Kirby		Mega Man	
baseline	rules err (train)	design err (test)	rules err (train)	design err (test)	rules err (train)	design err (test)
Blend	0.152	0.623	0.09	0.664	0.127	0.727
KNN	0.180	0.400	0.105	0.638	0.148	0.525
GA	0.143	0.537	0.103	0.672	0.143	0.592
Expansion	**0.134**	**0.345**	**0.08**	**0.536**	**0.118**	**0.502**

7.3.1 Evaluation

My ultimate goal for applying conceptual expansion over game graphs is for it to solve the video game invention problem, which requires the creation of novel games. However, it is difficult to evaluate the creation of novel games. Thus for initial evaluation purposes of this first variation, I present a scenario where the system learns about two games and must try to construct a third, closely related game. To allow for comparative errors, the third game is a previously extant game that is unknown to the system at the time of evaluation. While this is not ideal in terms of only representing p-creativity, it serves to give an initial sense of the performance of this variation along with giving a precise, objective evaluation measure.

I parsed two gameplay videos of the first level of three games: (1) Super Mario Bros., (2) Kirby's Adventure, and (3) Mega Man. I chose these three games as they are platformers for the Nintendo Entertainment System and they vary in terms of mechanics and design.

For example, Mega Man's levels move along the y-axis and include enemies with more complex behavior, impacting rules and level design. Further, Kirby's Adventure devotes a fourth of its screen to the game's UI, which also impacts the rules and level design. I only parsed the first level to reduce the size of the final game graph, given the complexity of this representation.

I ran two distinct experiments for this evaluation. For the first I gave the system two game graphs for its input, with the goal of a third game graph with only the level design model knowledge. This represents a hypothetical situation in which a designer on some game wanted to create the full game but only had a level design defined. In this experiment conceptual expansion search will attempt to create a whole game that matches this level design information, based only on feedback from the heuristic (e.g. with only a distance function to give relative feedback on the values of the goal game graph nodes). I can then compare both the final output game graph in terms of how the level design information differs (this is analogous to training error) and how the rule information differs (this is analogous to test error). For the second experiment I formulate the reverse situation, given a rules-only game graph recreate an entire game that matches both rules and level design knowledge. For example, given Super Mario Bros. and Kirby's Adventure game graphs, and with a heuristic that gives indirect feedback on how close a candidate game graph is to the rules of Mega Man, to what extent can conceptual expansion search (and the baselines I describe below) recreate the entirety of Mega Man.

I constructed three baselines for this evaluation. They are as follows:

- **Blend:** I applied conceptual blending to this problem, given its history in other automated game design systems. This can be understood as a subset of the search space explored by conceptual expansions and uses the implementation of conceptual blending discussed in Chapter 6, returning the output that best fits the same heuristic as the conceptual expansion approach. In particular, for each goal game graph node I take the top two mapped inputs from the mapping m and combine all of their edge

information to create final blended nodes.

- **KNN:** K-nearest neighbors represents one of the few machine learning approaches that can function with such a small dataset. This approach returns whichever of the input two game graphs is closest to the goal graph according to the distance function.

- **GA:** I constructed a genetic algorithm baseline given its application to other automated game design systems. I make use of a population of size 10, initially made up of the original two game game graphs and four mutations of each. The mutate function changes a random edge value to another value in the graph (e.g. sprite name to another sprite name, numeric value to another numeric value). The crossover function creates a new graph by randomly selecting half of the nodes of each parent graph. I make use of the same heuristic/evaluation function as in conceptual expansion search. This baseline was by far the slowest, taking between five and twenty times as long as conceptual expansion search with a cap of one-hundred generations.

A genetic algorithm (GA) is simply one way to optimize a set of parameters. One can think of conceptual expansion search in the framework of a GA as being a search only over the space of possible crossovers of the very first generation. Thus, I could apply a GA to optimizing the game graph's conceptual expansions, and would likely improve the results discussed below, given the greater optimization power of a GA compared to greedy search. However, I abstain from this to show the strength of the representation. The GA baseline therefore primarily demonstrates a different strategy for representing this problem.

7.3.2 Results

I present the results for the designer-experiments (given level design-only game graph as a goal, create level design and rules) in Table 7.1 and the developer-experiments in Table 7.2 (given rules-only game graph as a goal, create rules and level design). Conceptual Expansion outperformed the baselines in all but one case.

The developer-experiments appear significantly more challenging for all of the approaches. This matches my intuition that there could be many possible level designs for the same ruleset (e.g. there are multiple levels in a single game), while a particular level design model limits the kinds of rulesets that can successfully navigate the levels. For example, jump distances determine valid sizes of gaps. For the designer experiment with Mega Man as the goal the KNN baseline outperformed all approaches in terms of the rules (test) error. I anticipate the issue was that Mega Man's level design is not as tightly related to its rules, given that conceptual expansion and the GA baseline outperformed the KNN in terms of design (training) error. This follows from the fact that Mega Man receives a number of powerups that alter how he can move through the level, which different players gain access to at different times. Thus levels cannot be designed under the same assumption of what mechanics will be available as they can for Super Mario Bros. or Kirby's Adventure.

7.3.3 Discussion

These results demonstrate that conceptual expansion can be applied to create game graphs that more closely match a desired game graph given a distance to a partial specification of that game graph. However conceptual expansion search does not need a goal and could work with any appropriate heuristic, as I demonstrate in the next two variations. In these two variations instead of a distance to a goal I optimize for a heuristic that measures novelty, surprise, and value.

While the three games I chose as a dataset for this game differ from one another, they are all three platformer games originally published on the Nintendo Entertainment System, with two coming from the same company. Thus, I anticipate a need for a more complete study with multiple games from multiple eras of game design.

I evaluated this first variation in a simulated interaction with designers and developers on three classic games, and found that it out-performs state-of-the-art baselines on this task. This represents a sub-part the video game level invention problem that I refer to as the p-

creative video game level invention problem. For this problem it is clear that conceptual expansion outperforms combinational and non-combinational baselines consistently. While I do not include amalgamation or compositional adaptation, given the set structure from the goal graph they would have given the same output, which would have done approximately as well as the KNN baseline. To the best of my knowledge this system represents the first machine learning-based automated game designer.

This was a somewhat convoluted evaluation, but it speaks to the difficulty of converting creative domains to standard train-test paradigms. Later in this chapter I instead rely on human evaluation, which is more subjective but more straightforward.

7.4 Variation 2: Human-in-the-Loop, Heuristic-based Game Generation

In the prior sections I demonstrated how conceptual expansion out-performed conceptual blending, genetic algorithms, and K-nearest neighbors in recreating existing game graphs based on partial definitions of those existing game graphs. While the output game graphs can be considered novel from the perspective of the systems, given that they did not already exist in the system's knowledge base, the entire point of the experiments was to replicate existing games. The output game graphs were also not playable given that they were not converted from game graphs into playable pieces of software (despite containing all the knowledge necessary to represent a fully playable game). In this variation I alter the conceptual expansion game generation process to instead create novel, playable games through four changes.

First, I replaced the goal-based optimization problem from the prior chapter with a heuristic-based optimization, this heuristic was an extension of the metrics discussed in Chapter 6. The Chapter 6 metrics measured novelty and playability (as a stand-in for value) in combined level design models generated via combintional creativity techniques from pairs of input level design models. In this case I define novelty as the minimum of the asymmetric Chamfer distances from the novel game graph to each element in the system's

knowledge base of existing game graphs. I define playability in terms of the percentage of output level chunks from these combined level design models that could be traversed by an A* agent using an implementation of Super Mario Bros. mechanics. Both of these metrics were normalized to the range [0,1]. I adapt these metrics into a combined heuristic for novel game graphs by addition, thus the heuristic outputs values in the range [0,2]. Notably when a game was generated by this system, its associated game graph was included in the system's knowledge base of existing game graphs. In this way the novelty portion of the heuristic constantly pushes the system to produce new games. For playability I output five level chunks from five randomly selected L nodes for a particular game graph and determined the percentage of traverse-able level chunks given the set of rules from the novel game graph. This bakes in an assumption that the goal of any output game is to move from the beginning of a level to its end, but I find this a reasonable assumption given that this is true of all of the existing games (Mario, Kirby, and Megaman) in the system's knowledge base.

Second, I developed a technique to derive a set of Unity c# scripts from the mechanics defined by a novel game graph. Unity is a general purpose game engine [176], which is popular among human game developers for developing games. Unity's major benefit in this context is the ability to export playable video games across almost all platforms on which people play video games (e.g. personal computers, online, mobile devices, etc.). Unity uses a script-based system, in which in-game entities may have an arbitrary number of scripts attached to them and act according to the behaviors defined in these scripts. Thus this code allowed me to first covert a novel game graph back into the formal logic-esque rule representation, and then from this into a series of c# scripts.

Third, I added the level chunk transition information needed to sequence chunks of levels to each game graph. I did this via the addition of both an additional type of node without any sprites associated with it: the level chunk category node. I added one node of this type for each of the learned level chunk categories (derived via the process described

in Chapter 3) for each game. In addition I added four new types of edges associated with these nodes:

- **Level Chunk Type:** This simply stored the id of this level chunk category, which is an arbitrary but unique id generated during the level chunk categorization process. Notably this id is related to the information stored in **D**, **G**, and **N** edges to identify their associated L node. Thus generated L nodes can be associated with this level chunk sequence information when generating levels from game graphs. This edge is cyclic, pointing to the same component it originated from.

- **Level Chunk Repeats:** Stores a minimum and maximum number of times this level chunk type can repeat in a sequence. This edge type is cyclic.

- **Level Chunk Position:** Stores a float value specifying the average, normalized position this level chunk tends to be found in a sequence of level chunks. This information is not used during level generation, but is a helpful feature for mapping similar level chunk category nodes. This edge is cyclic.

- **Level Chunk Transition:** Stores a pointer to another level chunk category node and the probability of taking that transition. This information is taken from the learned level graph for each game, according to the process described in Chapter 3. However, notably this edge stores insufficient data to recreate the graph. Instead this solves the level chunk sequence generation problem with a Markov chain-like representation, sampling from possible transitions probabilistically until it hits a node without any outgoing transitions. This edge points from this node to the associated level chunk category node.

The inclusion of this level chunk transition information increased the size of the game graphs considerably, with the three existing game graphs now having hundreds of thousands of edges.

Fourth, I employed a human-in-the-loop generation process. Given the relatively simple heuristic I employed much of the knowledge about games represented by the knowledge base of existing games went unused. This lead to naive output from the system that included things like unkillable, immovable enemies, and powerups that went to the right at the same speed as the player (making it impossible to reach them). Therefore I curated what output game graphs were good and what were not. Game graphs that I judged as good output from the search process described in the prior sections, now leveraging the above heuristic, were re-represented in Unity and added to the system's knowledge base of existing game graphs. This is a limitation of this process, but was necessary at this stage to create playable output. The third variation addresses this limitation directly.

Further, because the components of a novel game were only represented as rectangles I also hand-colored each component to make them visual. I followed a relatively straightforward coloring rule, with the player character colored blue, main structure colored white, decoration colored green, and enemies colored red. I could have instead assigned a random color to each component, but thought this might impact readability. Given that all the existing games made use of a grid-based system I also employed a grid-based system in Unity, placing each component to its closest location on the grid. I named each game as well based upon my own subjective interpretation of its most identifying mechanic or level design. While rules in the game graph representation could be specific as control rules (in other words, rules that required player input), there was no way to specify what key or button should be associated with each control rule. Thus I came up with my own key bindings, following my own understanding of common practices in games. For example, rules that increased positive x-velocity were mapped to the right arrow, rules that increased y-velocity mapped to the up arrow and space bar, and so forth.

For an initial starting point for the greedy search process I used the Super Mario Bros. game graph. This notably represents an excellent starting point as it has high quality level design and mechanics as per the evaluations described in Chapters 3 and 4. I mapped each

node from the Kirby and Megaman game graphs to its closest node in the Super Mario Bros. game graph according to the asymmetrical Chamfer distance. This replaced line 1 of Algorithm 3. Thus the initial starting point (and therefore output games) were biased to be more like Super Mario Bros..

The entire pipeline for generating a novel game could then be expressed as follows. First, I took the initial conceptual expansion of the Super Mario Bros. game graph, combined with the Kirby and Megaman game graphs. Second, I optimized this game graph according to the greedy search described in the prior chapter and according to the heuristic defined above. Third, I checked the output game graph, verifying its quality. Fourth, if I found it to be of sufficient quality, I added the game graph to the knowledge base for the sake of the novelty comparison in the heuristic and translated the game into a Unity representation. I used the same Unity camera behavior for the output games of both this and the later variations. I made this choice for simplicity's sake and to ensure human players could always see the player avatar, which was not a given with this system.

7.4.1 Full Games Case Study

With the process described above I created two games, rejecting an additional three game graphs in the process. The first game I accepted, which was also the first game graph output by the system, is in Figure 7.4. I nicknamed this game "Death Walls", due to the extremely tall enemies. These tall enemies were a combination of the Goomba enemy from Super Mario Bros. and tall rod-lick decorations from Megaman. These nodes ended up mapped onto one another due to a rule learned while the engine learning system was parsing the Megaman video. Due to their size the tall decoration appeared to disappear when Megaman stood in front of it, leading to a rule being learned that destroyed this decoration when it collided with Megaman. This rule is incorrect, but it lead to this interesting mapping and created the final tall enemy component in this generated game.[1]

[1]There is video of me playing this game available at https://www.youtube.com/watch?v=urKsby4-AbQ.

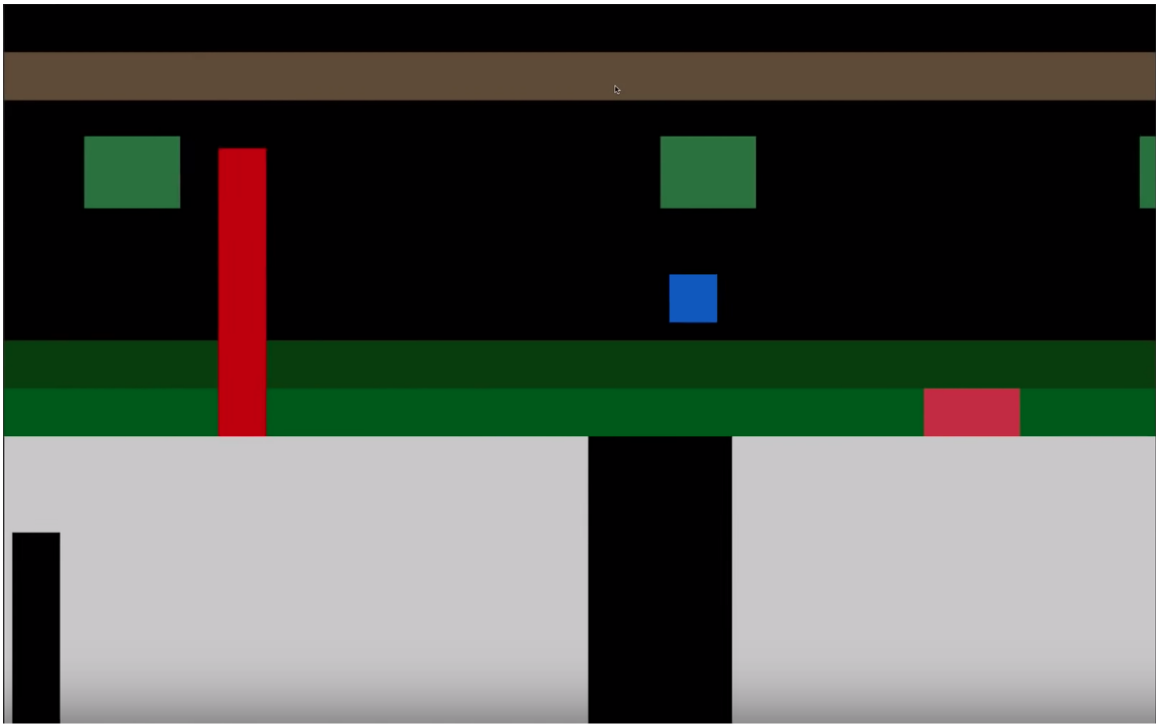

Figure 7.4: A screenshot taken from the first game output from this process, which I named "Death Walls".

The second game output from this process took longer, with three output game graphs rejected before it was produced. Thus it was the fifth game graph in total. I include an illustrative screenshot of the game in Figure 7.5.[2] This game has an unusual mechanic in which every time the player hits any structure it was destroyed, thus I named the game "Killer Bounce". In the final "Killer Bounce" game graph there was one node that was a combination of the Goomba, Ground, and Brick components of Super Mario Bros., reached due to a series of random operators that added the second two f values to the combined Goomba node and then altered the a values of this combination. This lead to both the main mechanic of the game and level structure.

The primary mechanic arose due to a rule that destroyed a goomba (set it's animation fact to $None$) if the player was just above it ($RelationshipY = 0$). This is actually technically an incorrect rule as the Goomba should have simply transititioned to a different animation state and then disappeared, but its functionally correct. The primary level de-

[2]Video of me playing the game can be found at https://www.youtube.com/watch?v=PX5ig7r3G6U.

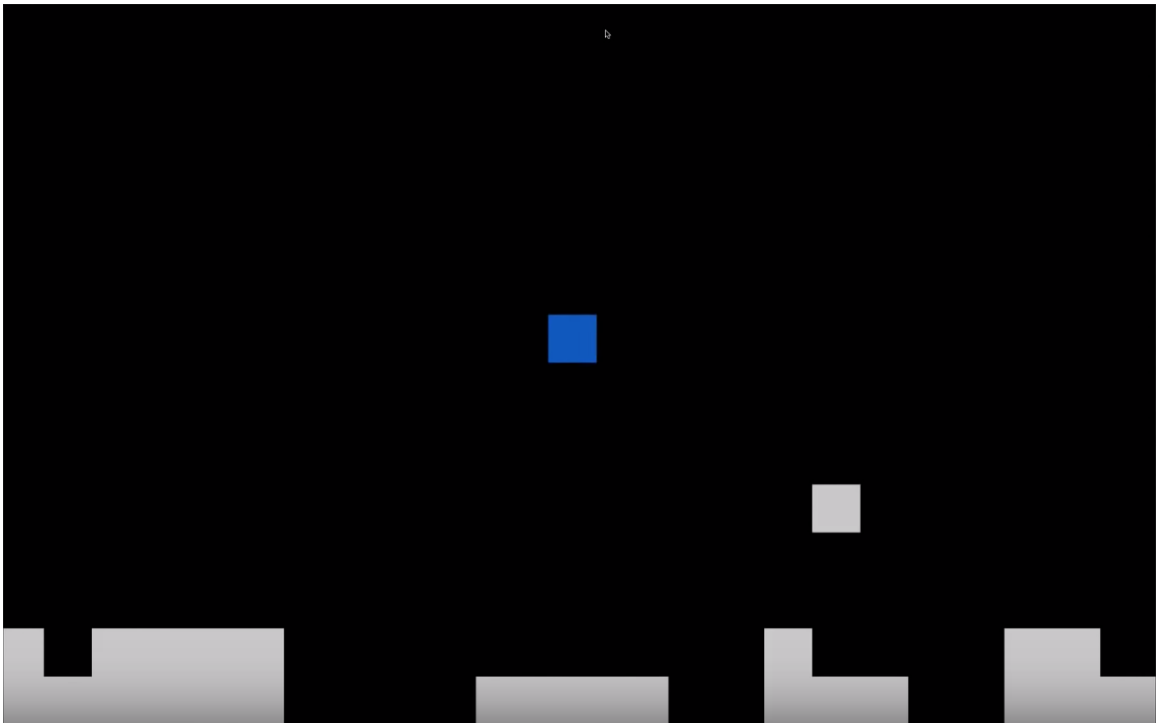

Figure 7.5: A screenshot taken from second game output, which I named “Killer Bounce”.

sign of “Killer Bounce” involves floating staircases of blocks and somewhat noisy ground, this arose from the combination of f values as well, with the Brick leading to the floating staircase-like structure and the Goomba and Ground leading to the noisy ground. The final component takes the relationship between ground and goombas (goombas tend to be placed just above the ground) and instead places two differently sized shapes of itself, one atop the other.

7.4.2 Discussion

In this case study I introduced the first completely novel games created via a machine learning-based automated game designer through the application of conceptual expansion. As with the prior variation I cannot consider the system to have addressed the video game invention problem, in this case due to my own involvement in the game production pipeline. However, given the nature of this pipeline, both of the output games reflect elements in the potential output space of conceptual expansion. Because these game graphs existed within

the output space of conceptual expansion, they are games that could have been output by an entirely artificial system, outside of their names and visuals. However, due to my involvement a formal evaluation of these games would be meaningless, as it would not be possible to differentiate what results were due to my involvement versus the conceptual expansion approach.

There are several elements lacking or that require alteration for this process to be meet the requirements of a solution to the video game invention problem. First, I need to remove the human from the process as much as possible in order to make the argument that the success of the output comes from conceptual expansion. This can be achieved by improving the heuristic to remove the need for a quality check. Second, I need to produce fully playable games for a more general audience. This requires having more readable visuals without someone having to assign each entity a color. I address both of these points in the third variation.

7.5 Variation 3: Spritesheet-based Game Generation

One of the major drawbacks the second variation is the lack of visuals. Given that the system only understands components of games as rectangles of a particular width and height (in terms of the values in G nodes and Animation facts), there is no inherent way to express any novel output games visually. Therefore I made use of an open source spritesheet.

It is common practice for artists to make spritesheets for games that do not exist. Some artists open source these spritesheets to allow game designers to draw on them for making new games. If I could alter the conceptual expansion generation of game graphs to target a particular spritesheet and make a game for it, then that would give that game visuals without the need for human intervention. Further, this limits the problem from one of creating any possible video game to creating games that match a particular spritesheet, making the evaluation much simpler. I could have generated or modified an existing spritesheet in order to have an additional avenue for artificial creativity, but I focused in this work on non-visual

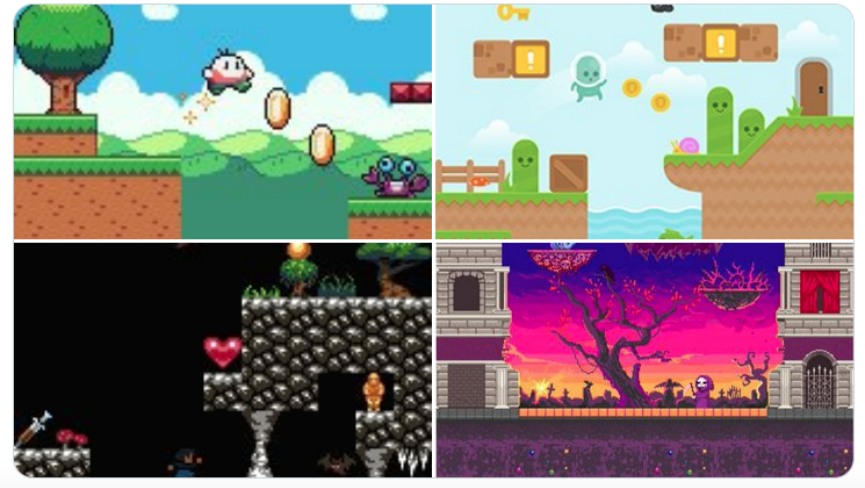

Figure 7.6: A screenshot of game-like screenshots for the four spritesheets options for the spritesheet-based game generator and following study.

aesthetic elements of game design. Further, the work in automated spritesheet generation is still too immature for application in a complete game generation system [177].

In this section I cover the pipeline for the spritesheet-based conceptual expansion game generator, in terms of the choice of spritesheet, the construction of what I call a proto-game graph from that spritesheet, the mapping from the three existing game graphs onto that spritesheet, and the heuristic I designed to represent value, surprise, and novelty.

7.5.1 Choosing a Spritesheet

Any spritesheet chosen in this process would constrain the possible output games for the conceptual expansion game generator. It would be theoretically possible for me to intentionally choose an open source game spritesheet that would benefit conceptual expansion and this overall game generation process, given my familiarity. For example, choosing a spritesheet with the same number of components as one of the output games from the second variation, such as "Killer Bounce". For this reason I chose instead to leave the spritesheet up to public vote.

I collected four appropriate spritesheets and posted them to social media to solicit feedback as in Figure 7.6. I determined appropriate spritesheets based on those that could po-

tentially represent a platformer game, had all aspects of a game (not just a player's sprites or ground sprites), were open source, and were relatively popular (e.g. at least one game existed using the spritesheet). The four chosen spritesheets were "Arcade Platformer Assets" by the artist GrafxKid from opengameart.org (top left), "Platformer Art Deluxe by the artist Kenney from his website kenney.nl (top right), "Simple broad-purpose tileset" from the artist surt from opengameart.org (bottom left), and the 2D Art Pack by the artist Cryoclaire for Procjam 2017 [178] (bottom right).

I ran a poll asking people to vote on which of these they would prefer if they had to play two games in a row with the same visuals. After twenty-four hours, I collected ninety-nine votes. The first option "Arcade Platformer Assets" by GrafxKid took a majority of the votes, with "Platformer Art Deluxe" by Kenney coming in second place.

I have included the entirety of the GrafxKid spritesheet in Figure 7.7. I also include a simplified version of the Kenney spritesheet in Figure 7.8 as I used it as the spritesheet during development of this spritesheet-based conceptual expansion game developer in order not to bias the development towards the GrafxKid spritesheet. Thus I also use the Kenney spritesheet in an example below while describing the pipeline.

7.5.2 Proto-Game Graph Construction

For the first variation I demonstrated how with a partially specified goal graph, conceptual expansion could fill in the gaps to recreate an existing game graph. It stands to reason that constructing a game graph-like representation from the spritesheet could prove helpful in this process, at minimum allowing me to utilize the same distance functions from the previous chapter for mapping game graph components. Towards this end I developed a process to construct what I call a "Proto-Game Graph" from the spritesheet.

It is common practice for visually similar sprites in a particular spritesheet to be related in terms of game mechanics and/or level design. For example, take the pink astronaut creatures from 7.8, it is clear that these sprites were likely intended as different animation

Figure 7.7: The entire GrafxKid “Arcade Platformer Assets” spritesheet.

states for a single in-game entity.

Given this intuition I employ one round of single-linkage clustering on the individual sprites of the spritesheet in an attempt to automatically group visually similar sprites. First, I represented each sprite as a bag of all of its 3x3 pixel features, given that bag of features strategies tend to perform well on image processing tasks even compared to state-of-the-art methods [179]. I then constructed a simple distance function represented as the size of the disjoint set of features divided by the maximum size between the two compared bags of features. After running a single iteration of single-linkage clustering the Kenney spritesheet looks like 7.9. As one can see, while there’s some unusual links (for example

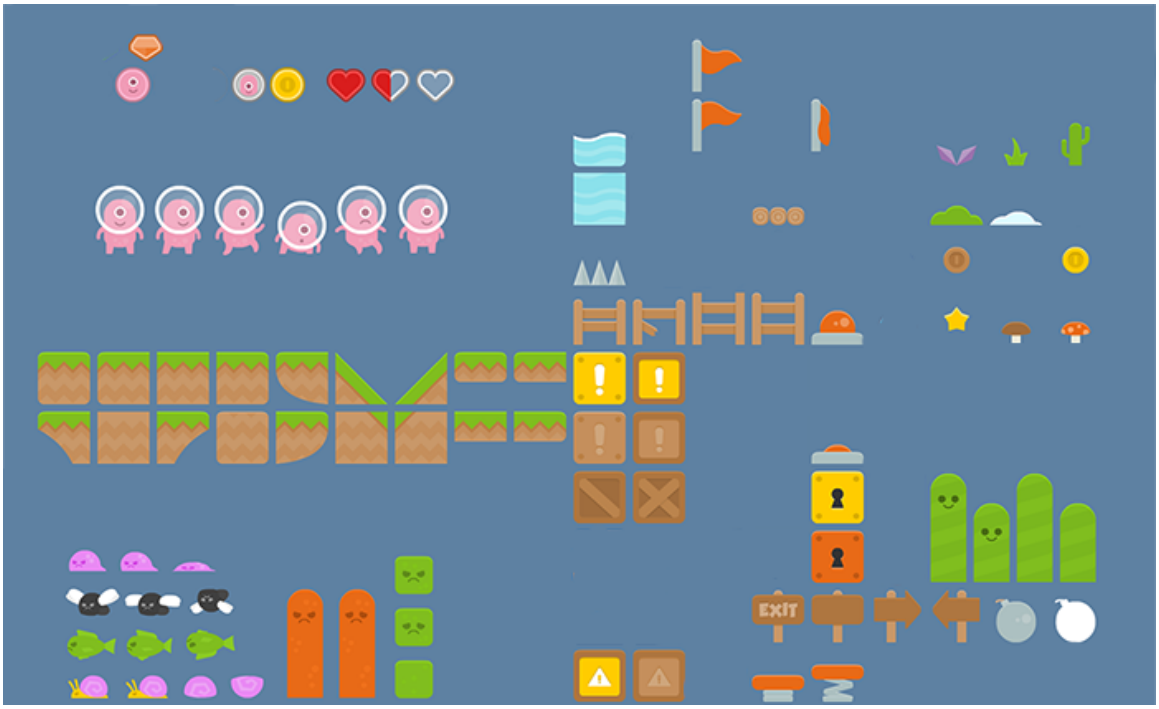

Figure 7.8: The simplified Kenney "Platformer Art Deluxe" spritesheet.

the grass and cactus being connected to fish), overall these groups reflect basic intuition about the visually similar sprites.

I treat all clusters as a single game graph node. Thus, for example, there would be one game graph node for all of the pink astronauts cluster, one for the cactus, grass, and fish cluster, and so forth. Each of the individual sprites in the cluster is then translated into an animation fact edge and a G node edge. This gives an initial, underspecified game graph that I call a proto-game graph. The proto-game graph technically has some mechanical (anaimation fact) and level design (G node) information, but does not represent a playable game and could not be used to generate levels. However, by having it in the same representation as a true game graph I can treat it as one for the purposes of constructing a mapping.

As with the three other game graphs I also identified the game graph nodes that should be used as the player of the game, represented as a simple boolean value on each node. This was necessary as the player is identified for the three existing game graphs and due

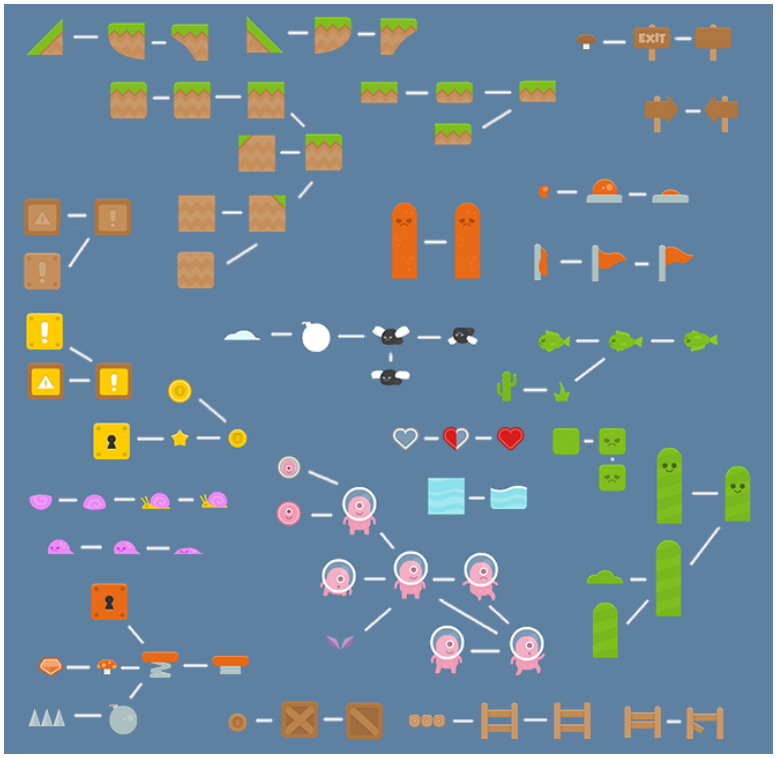

Figure 7.9: The result of running single linkage clustering for a single iteration on the Kenney spritesheet.

to the heuristics at play. However, in order to minimize the potential for me to impact the creativity of the output all the baselines (including human-designed games) were also informed what sprites to treat as the player.

7.5.3 Proto-Game Graph Mapping

The first step of any combinational creativity approach is to determine a mapping. In this case the nodes of the existing game graphs need to be mapped onto the nodes of the proto-game graph. To accomplish this each node of all three of the existing game graph was mapped onto the closest node in the proto-game graph. Notably this was still using the asymmetrical Chamfer distance metric, which meant that it didn't matter that the proto-game graph nodes were under-specified. This distance function varies from [0,1] and I only allowed mappings where the distance between the two nodes was <1, thus ensuring they had to have some similarity. This mapping left some nodes in the proto-game graph

without any existing game graph nodes mapped to them, which would have left them empty and meant that they had no chance to appear in any generated games. To rectify this I mapped each of these empty proto-game graph nodes to its closest existing game graph node (basically flipping the direction of the distance function).

This process also left some existing game graph nodes unmapped due to having no similarity with any of the proto-game graph nodes. In particular these were the newly introduced level chunk category nodes, nodes pertaining to the camera, and to the concept of nothing ("None") used to represent empty game entities in the ruleset learning process. However, these nodes contained necessary information to transform the proto-game graph into a game-graph capable of generating levels. Therefore I added nodes for the Camera and None entities to the proto-game graph, due to the fact that all of the existing game graphs also had one of each of these specialized nodes. For the remaining unmapped existing game graph nodes I ran a K-medians clustering with k estimated with the distortion ratio [153]. For the GrafxKid spritesheet proto-game graph this lead to five clusters of level chunk category nodes. I added all of the information from these clusters to a single node, meaning a final five level chunk category nodes appeared for the GrafxKid spritesheet proto-game graph.

This mapping was used to derive an initial conceptual expansion for the conceptual expansion search process. It was also used as the mapping for the three combinational creativity baselines (amalgamation, conceptual blending, and compositional adaptation) that I used for the human subject study described below.

7.5.4 Heuristic

For this third variation I began with the same heuristic I employed for the second. I altered the heuristic for this spritesheet-based game generator in order to remove the need for a human-in-the-loop. Further I wanted this heuristic to reflect the three aspects of creativity identified by Boden [17]: novelty, surprise, and value.

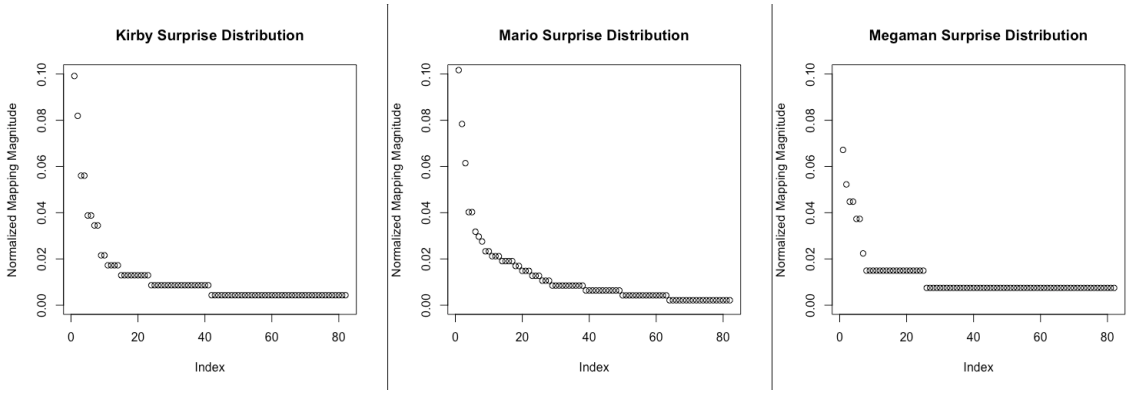

Figure 7.10: Surprise distribution comparisons for the three existing games.

For novelty I employed the same metric as discussed for the second variation: the minimum distance comparing the current game graph to all existing game graphs in a knowledge base. The knowledge base in this case again includes the three original existing game graphs, and any generated game graphs created by the conceptual expansion game generator. Notably in this case I do not include the game graphs from the second variation and do not intervene to determine appropriate game graphs. Intuitively this can be understood as a measure of how dissimilar a particular game graph is compared to the most similar thing the system has seen before.

Surprise is difficult to represent computationally given that it relies on some audience's expectations being contradicted [17]. Notably I also wanted a metric that wouldn't just correlate exactly with novelty, but would allow me to represent distinct measures. Thus I first built a representation of audience expectations by constructing what I call *Normalized Mapping Magnitude Vectors*. One of these vectors is constructed for each of the three original game graphs, by mapping each node of each game graph to its closest node in the two remaining graphs. From this I collected the number of times each of the remaining two graph's nodes had been mapped, sorted these values and then normalized them. This gives the one-tailed distributions in Figure 7.10. These essentially reflect how a person only seeing one of these games would relate it to the games they had already seen, thus this hypothetical person's expectations. To determine the surprise for a novel game graph the

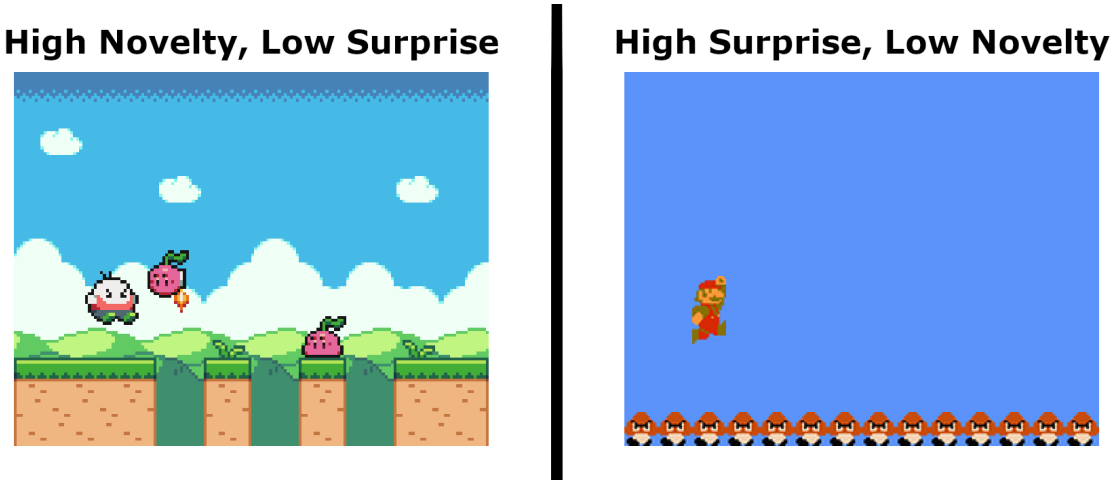

Figure 7.11: Comparison of two theoretical games that would receive differing novelty and surprise values according to the heuristic.

system constructs the same distribution by finding these mapping counts from the current graph to these existing graphs (including any graphs produced by the system). I compared the distributions by cutting the tails so that they were equal, normalizing once again, and then directly measuring the distance between the two lines at each position. Given these were normalized vectors, the max difference between the two would be 2, and thus I divide this number by 2.0 to determine the surprise value compared to each existing game. I then take the minimum of these values as the final surprise value.

The surprise metric may seem unintuitive, as such I include an example of two theoretical games, visualized in Figure 7.11. On the left side of Figure 7.11 is a game that would give high novelty as each component is individually fairly different from its closest equivalent in any of the existing game graphs. However, the mapping to the closest equivalent node across all the game graphs matches the distributions seen in Figure 7.10 as there is a player, enemies, and structure just like these existing games. Comparatively on the right is a game with only two components, meaning that it's normalized mapping magnitudes vector would be (0.5, 0.5), which differs significantly from the closest equivalent of the existing games. However, it's novelty would be very low as it exactly uses two existing components from an existing game graph (Super Mario Bros.).

Value is the final creativity component to represent in this heuristic. As expressed above, value in games is difficult to evaluate objectively [180, 181]. Given the focus of this document is not on this unsolved problem I instead focused on only one aspect of a video game's value, which is more easily represented computationally: challenge. A game should not be too challenging or too easy, this is the basic concept behind the notion of flow [182], which is a commonly applied as an approximation for quality in video games [180]. In the second variation I focused on whether level chunks output by game graph were traverseable by an A* agent, but this does not reflect the challenge or difficulty of traversing that level chunk. However, the notion of automatically reflecting human experience across a level is another non-trivial problem and a current research area in terms of *automated playtesting* [83, 160, 162].

Instead I kept the A* agent simulation approach from prior versions of this heuristic. However, instead of only using the coarse measure of whether or not the agent could complete a level, I collecting a three summarizing statistics from the A* agent's search across the level chunk. Specifically I collected the max distance the A* agent traversed, normalized to the width of the level chunk (0 meaning the agent never moved, 1 meaning the agent made it to the end), the number of times neighbors to a current node did not include the A* agent (the agent was destroyed) and the number of times the A* agent fell below the level chunk's lowest point (falling off screen).

I collected these summarizing statistics across one-hundred simulations of the original three game graphs. This allowed me to derive a distribution over these values for the original games. However, given the time cost in simulating an A* agent, when the heuristic was called I did not wish to exactly simulate one-hundred level chunks for each game graph. Instead I simulated only five level chunks and compared the median, first quartile, and third quartile of this five sample distribution and the existing one-hundred sample distributions. In this case the goal for the system is to minimize the distance between this representation of value and the original games (as the original games are assumed to have been designed

with appropriate challenge). Thus if the median, first quartile, and third quartiles exactly match any of the three original existing games this returns a 1.0, with the linear distance taken otherwise. Notably this means that the value metric is the only metric that does not involve any generated games previously output by the system, as there is no way to ensure the previously generated games are well-designed in terms of challenge.

Taken together then the maximum heuristic value is 3.0, with a maximum of 1.0 coming from each of the novelty, surprise, and value metrics. An ideal game then would be nothing like any of the original input games or previously output games, while still being roughly as challenging.

7.5.5 Human Subject Study

In this section I discuss the human subject study that I designed to evaluate this third variation, the spritesheet-based, conceptual expansion game generator. I first discuss the nine games created for the study in terms of the three types of generators of these games (human game designers, my conceptual expansion approach, and the three existing combinational creativity approaches). Then I discuss the process participants went through in the study, in which they played three games in a row and took a post-survey.

Human Games

I contacted seven human game designers to make games for this study. I asked these designers to create a game under the following constraints, meant to level the playing field as much as possible and ensure all parties were working on the same task:

- The game should be a platformer, but this could be "interpreted however [the author] liked". One game designer interpreted this to be about making jumps, one about collecting coins, and one about precision movement.
- The games could only use visuals from the GrafxKid spritesheet 7.7, with the white, round creature being the "basis of the player".

- The game should restart immediately upon player death. This constraint was added as the non-human generated games would simply stop if the player died or disappeared unless I specifically made the game restart.

- The game must not have music or sound effects. This was included as the current game graphs include no audio data.

- The game can only make use of arrow keys and spacebar as input for player control. This was included to minimize the risk of bias in using these keys to handle control rules in generated games.

- The ending of the game "should be obvious". For the generated games I simply froze the game when the player reached the end, but did not want to limit the human game designers to this, given that some of the tools they used to build the games made this more challenging.

- You must spend less than eight hours on the game, but you may spread these eight hours over any length of time. This constraint was meant to keep the games produced by the humans at roughly a "prototype" level of quality, given that the non-human designers had no way to iteratively improve the games.

- The games must be playable online.

Three designers agreed to take part, each producing one game. I contacted two of these designers through Chris DeLeon, who took part in this study. Chris DeLeon is the founder of Gamkedo Club, and the two remaining designers were members. Gamkedo Club is a mentored practice community for game developers with members worldwide collaborating online. Each designer was paid 60.00 USD upon agreeing to take part and 60.00 USD afterwards, for a total of 120.00 USD. This value was chosen as it represented 15.00 USD per hour if the designers took the maximum amount of time. Due to my involvement in this process, a separate video game playing expert without knowledge of this study outside

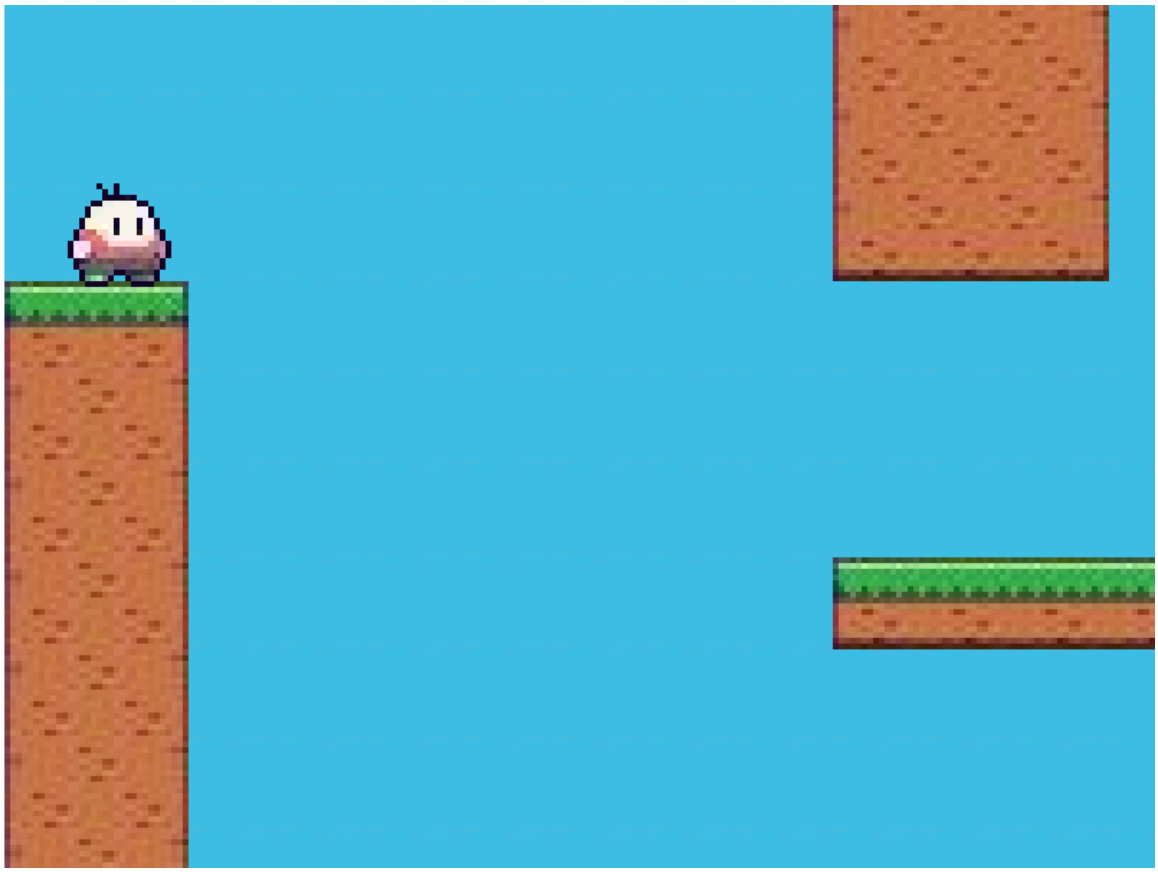

Figure 7.12: A screenshot taken from the beginning of Kise's game.

of these contracts verified the games were sufficient and met the constraints listed above. Notably, I did not see the human-made games until the study launched. I highlight each designer and their game in the order in which the games were completed below. Information about the game designers and games was collected after the designers submitted their games and had them approved by the third party.

- **Kise (K):** The first game designer goes by Kise and described herself as a hobbyist game developer. I have included a screenshot taken from the beginning of her game in Figure 7.12. She developed her game in javascript, and took inspiration from the game Super Meat Boy. She only made use of the ground sprites, a single player sprite, and a sparkle to signify the end of the level. The player retains acceleration in the x-dimension, leading to a "slidey" feeling. The game is playable at http://guzdial.com/StudyGames/0/.

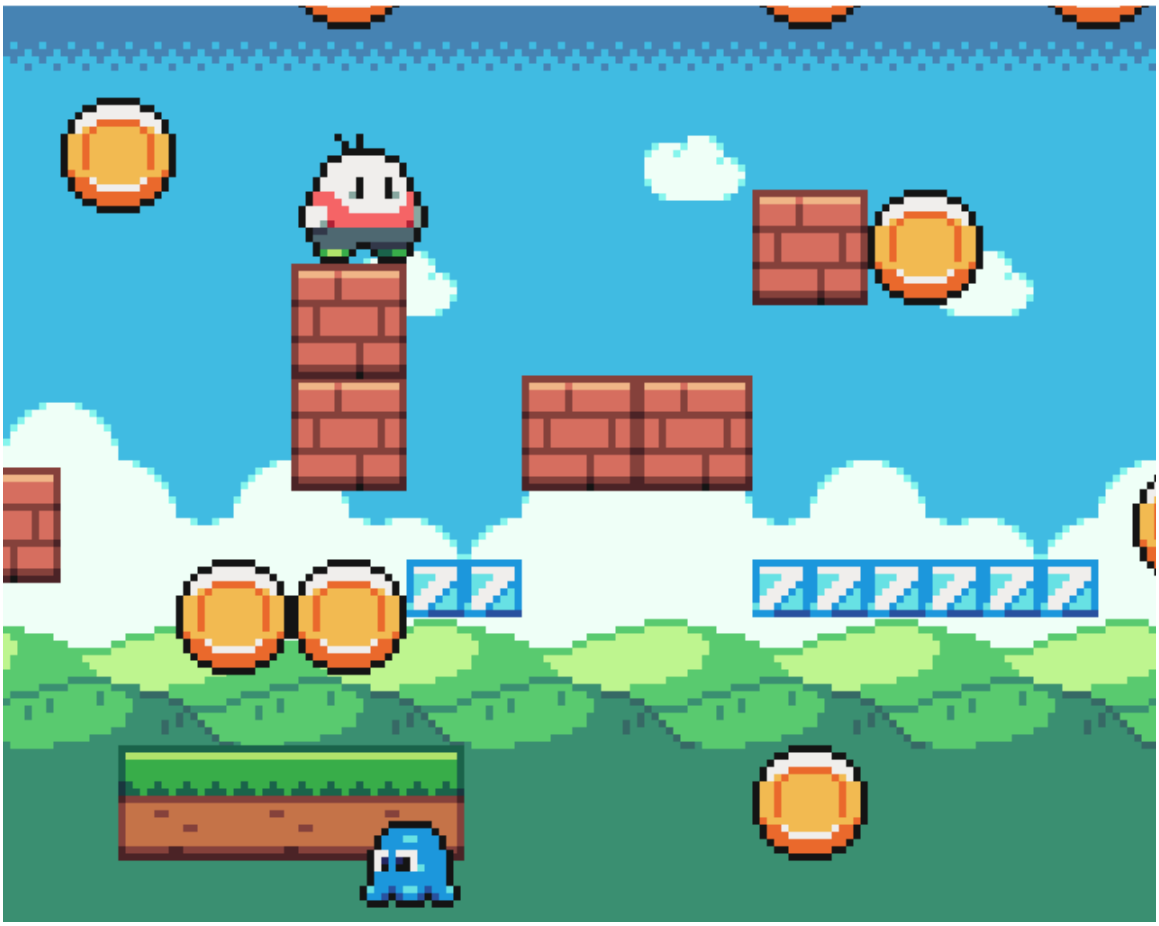

Figure 7.13: A screenshot taken from the beginning of Chris DeLeon's game.

- **Chris DeLeon (CD):** The second game designer was Chris DeLeon. He described himself as an "online educator". His game was also developed in javascript, and I include a screenshot of it in Figure 7.13. This is much more of a puzzle game, with the goal being to collect all of the coins (some hidden in the blocks) without falling off the screen or without the blue creature touching you. Again there were no animations for the player character, but there were for the blue enemy creature. Notably the player cannot jump on the icy blue platforms, meaning that it is possible to become trapped in an unwinnable state. One can find more from Chris DeLeon at ChrisDeLeon.com. The game is playable at http://guzdial.com/StudyGames/1/.

- **Trenton Pegeas (TP):** The third game designer was Trenton Pegeas. When asked to describe himself he stated he'd "like to classify as indie, but closer to hobby-

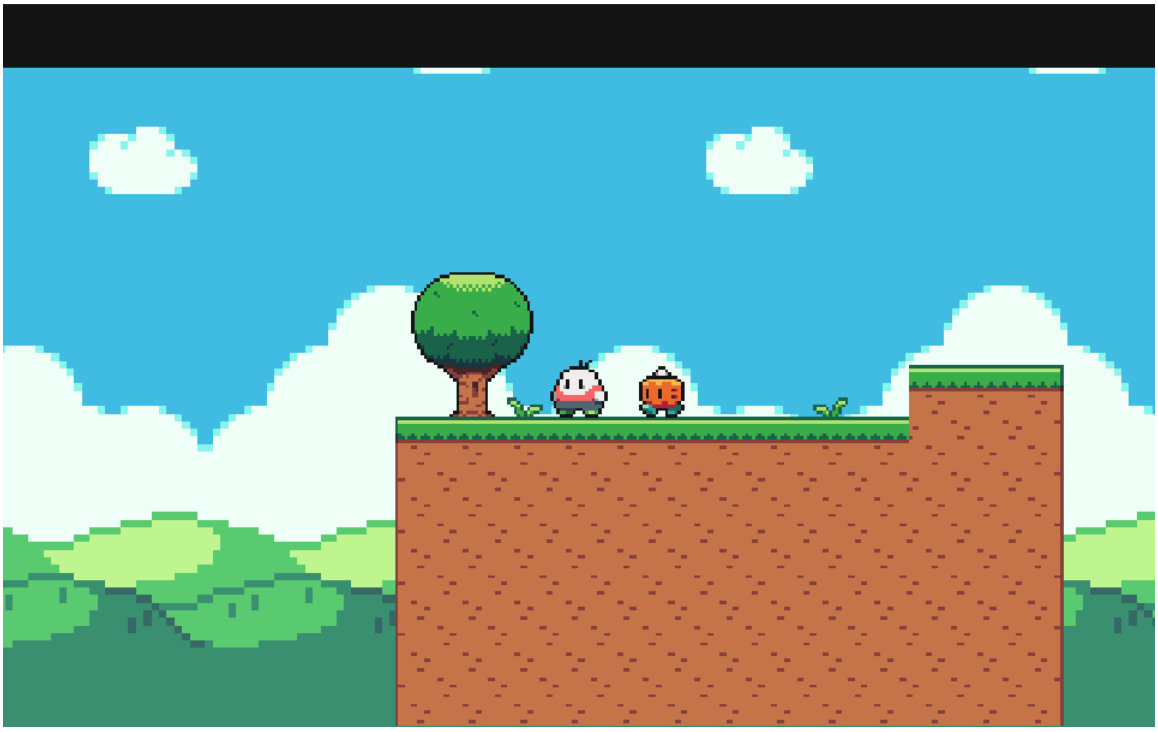

Figure 7.14: A screenshot taken from the beginning of Tenton Pegeas' game.

ist". Unlike the other two, he created his game in Unity. He also created by far the more complex game, both in terms of creating the largest level and including the most sprites and entities in his game. The player character and enemies all animation using all of the available sprites from Figure 7.7. The player follows a fairly linear path winding along the level from their starting position, interacting with different entities until they make their way to a goal. One can find more from Trenton Pegeas at https://twitter.com/Xist3nce_Dev. The game is playable at http://guzdial.com/StudyGames/2/.

Conceptual Expansion Games

I generated three conceptual expansion games (shortened to "expansion games") with the conceptual expansion search as described in Algorithm 3, but employing the heuristic described above. Specifically, from the initial starting point defined by the mapping, the system samples ten random neighbors from the space of potential conceptual expansion

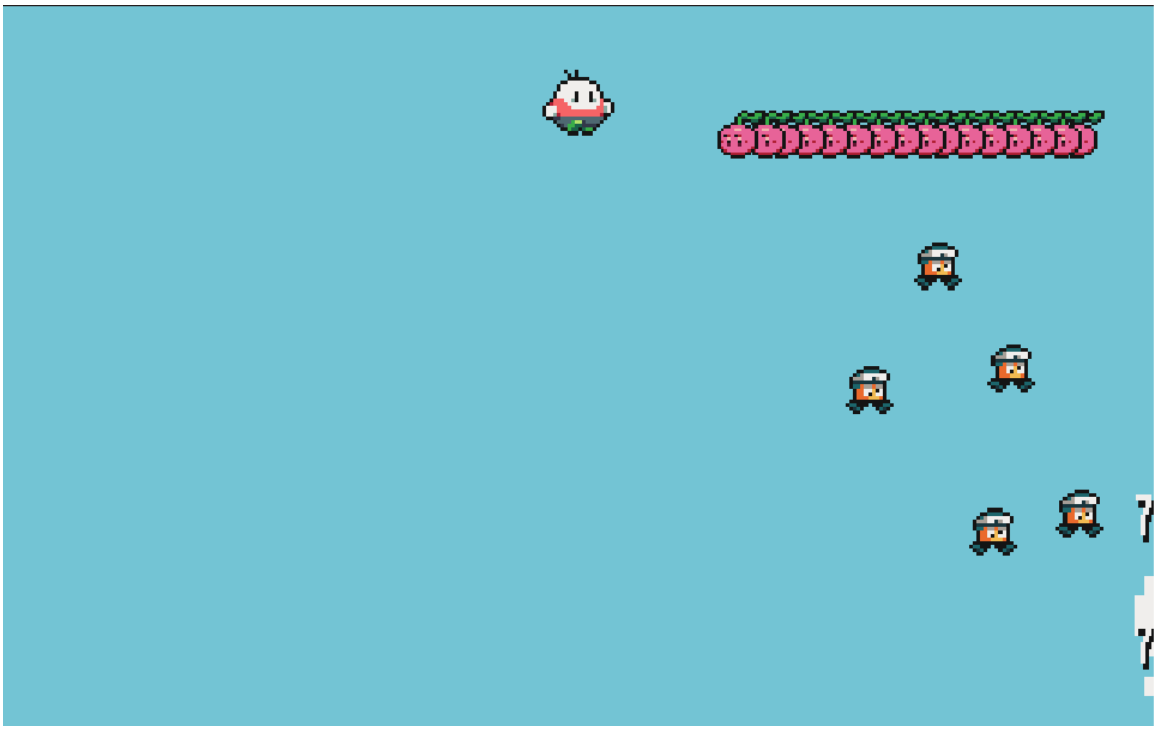

Figure 7.15: A screenshot taken from the beginning of the first expansion game.

output. If a neighbor has a better score in terms of the heuristic, it becomes the current conceptual expansion game graph. If the search samples all ten random neighbors without finding an improvement it assumes it reaches a local maxima and returns the current conceptual expansion game graph.

After each conceptual expansion game graph was generated it was added to the knowledge base of existing games for the system, which impacted the novelty and surprise portions of the heuristic. Thus the second game had the first game in its knowledge base and the third game had both the first and second games in its knowledge base. Below I briefly describe all three games in the order they were generated.

I could not completely remove myself from the process, as I had to copy and paste the output scripts onto appropriate components in the Unity game engine and choose key input mappings from the ones available to the human designers. However I contend that this is a minimal intervention that is unlikely to impact the final perceived quality of the games. Further, I had to do the same for all the non-human generated games, both these

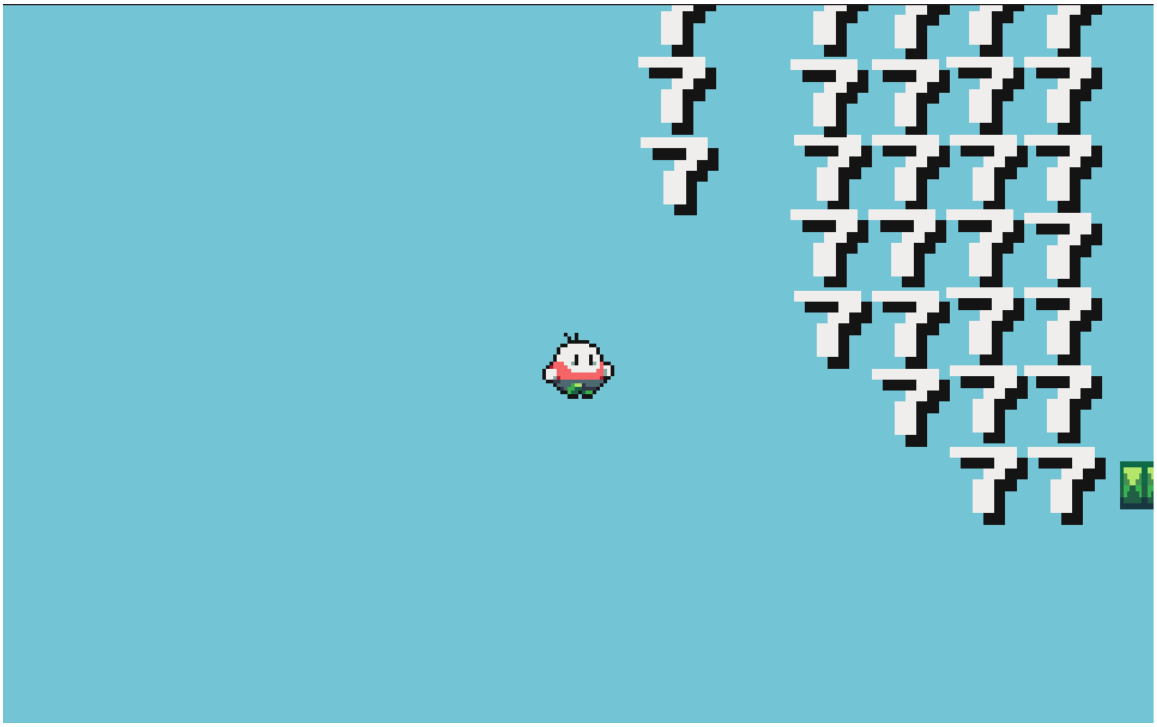

Figure 7.16: A screenshot taken from the beginning of the second expansion game.

games and the games output by the three existing combinational creativity approaches. For a background color I chose a single off-blue color from the spritesheet, which I used for all non-human games. I also got rid of the grid system used by the second variation, as it unnaturally constrained the output to be more like the knowledge base games.

Notably the expansion games outperformed the combinational creativity baseline games in terms of the final heuristic values for the novelty metric, which is surprising given that they had additional points of comparison. This is likely due to the larger output space of conceptual expansion.

- **Expansion Game 1 (E1):** The first expansion game generated by the system, and therefore notably the only one based solely on the existing games, is not much of a platformer. Instead the system produced something like a surreal driving or flying game, with the player moving constantly to the right and needing to avoid almost all in game entities since the player dies if it touches them. The player avatar animates slightly when the player moves up or down, but otherwise remains unchanged.

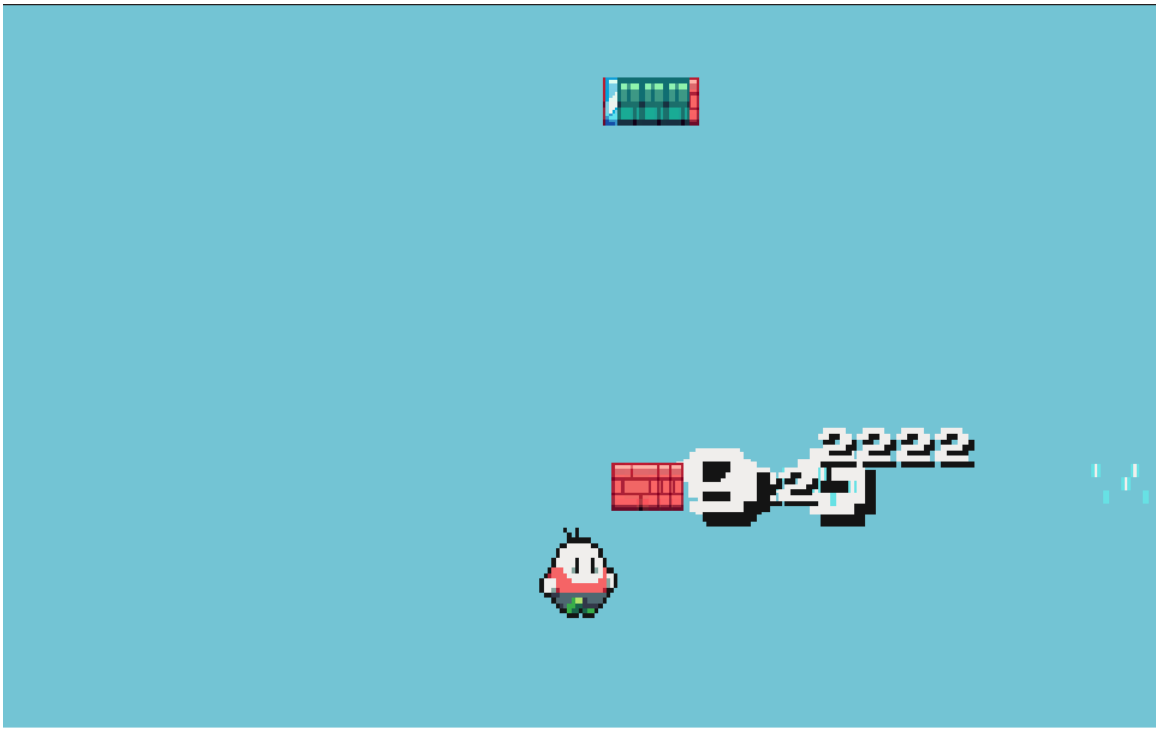

Figure 7.17: A screenshot taken from the beginning of the third expansion game.

Nothing else animates. The game is playable at http://guzdial.com/StudyGames/3/.

- **Expansion Game 2 (E2):** The second expansion game, which included the prior game in its knowledge base for the novelty and surprise measures, was also not much of a platformer. If the player moved left or right they would slowly go in that direction while gently falling along the y-axis. If the player went up they shot up quickly, meaning that the player has to balance going up to a set rhythm, somewhat like the game Flappy Bird. If the player went too high or too low they died and if they stood still they died. There is nothing to dodge in this game, meaning that once the player gets the rhythm of hitting the up arrow or space bar it is a simple game. The player avatar animations slightly when it goes up but not otherwise. The game is playable at http://guzdial.com/StudyGames/4/.

- **Expansion Game 3 (E3):** The third expansion game, which included the prior two games in its knowledge base for the novelty and surprise measures, was even more

like Flappy Bird. I include a screenshot in Figure 7.17 The player normally falls at a constant speed. The avatar moves slowly left or right when the player hits the left or right arrow. If the player hits the up arrow or space bar the avatar shoots up, with the inverse happening if the player hits the down arrow. The game is largely the same as the second game except for the more surreal level architecture and an area of "anti-gravity" (as dubbed by a study participant) whenever the player is above or below a patch of heart sprites. The player stretches when they go up or down, but nothing animates otherwise. The game is playable at http://guzdial.com/StudyGames/5/.

Baseline Games

To determine the extent to which conceptual expansion was better suited to this task compared to existing combinational creativity approaches, I included one game made by each of the three combinational creativity approaches highlighted throughout this document (amalgamation, conceptual blending, and composition adaptation).

In all cases I employed the same process to derive these games as the conceptual expansion search, only instead of sampling from the space of possible conceptual expansions I instead sampled from the particular output spaces of each approach. The only change I made to these existing approaches was in allowing for more than two inputs, but given that all three approaches employed the same mapping as the conceptual expansion I largely side-stepped this issue as most nodes in this mapping were mapped to a single existing node. I chose to include a single game from each approach as each approach has been historically designed to only produce one output for every input. I briefly highlight each game below.

- **Amalgamation (A):** The amalgamation process meant that whole existing game graph nodes were used for each node of the proto-game graph. This lead to perhaps the second most platformer-like of the generated games, with the player largely having the physics of kirby from Kirby's Adventure (as understood by the Engine

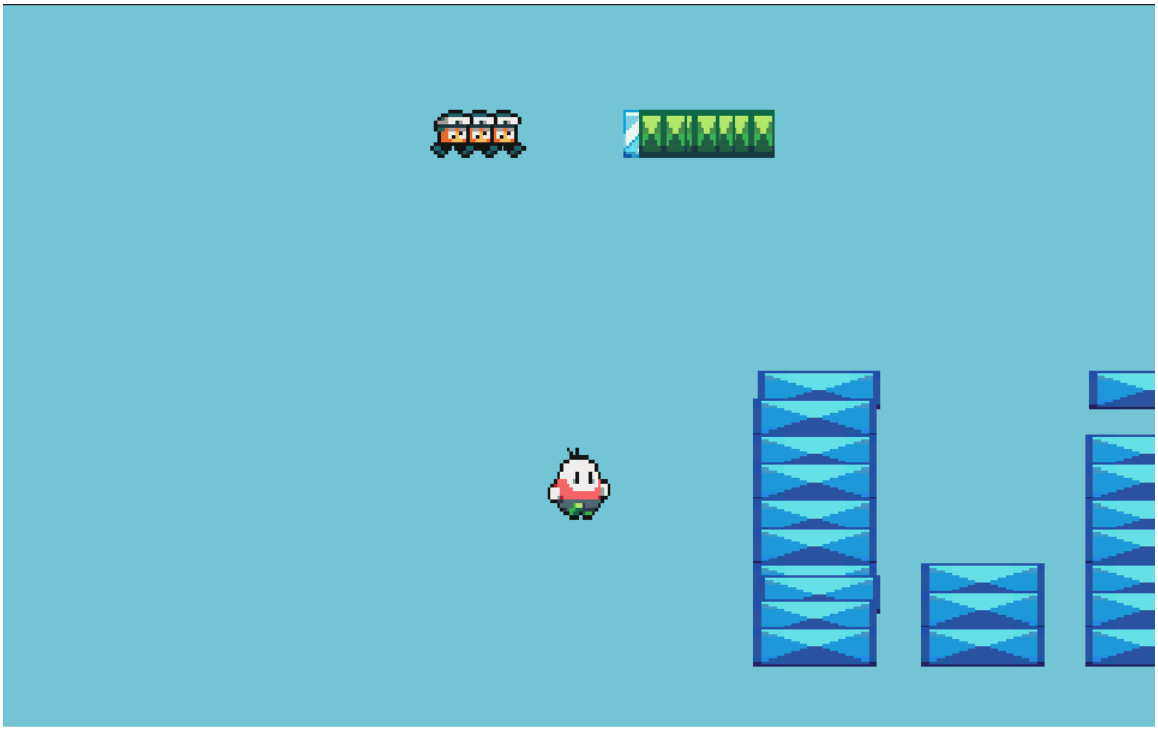

Figure 7.18: A screenshot taken from the beginning of the amalgamation game.

learning system), though slightly broken due to some nodes from the Kirby game graph not being included. A screenshot of the game can be seen at Figure 7.18. Perhaps the oddest part of this was the inclusion of blue columns that slowly drifted upward at the start of the game. This was due to a mapping to a decorative element from the Kirby games which the engine learning system incorrectly learned moved upwards. Because of the Kirby-esque physics it is almost impossible to lose this game. The game is playable at http://guzdial.com/StudyGames/6/.

- **Conceptual Blending (B):** The conceptual blend combined as much of the mapped existing game graph nodes as possible for each node of the proto-game graph. This lead to a game strikingly similar to the second and third expansion game, except with the notable difference that at the end of every frame the player transformed into the crab arm sprite as seen floating in the middle of Figure 7.19. Thanks to a serendipitous mapping that lead to treating the crab sprite as ground this lead to a surprisingly semantically coherent (if strange) game. The game is playable at

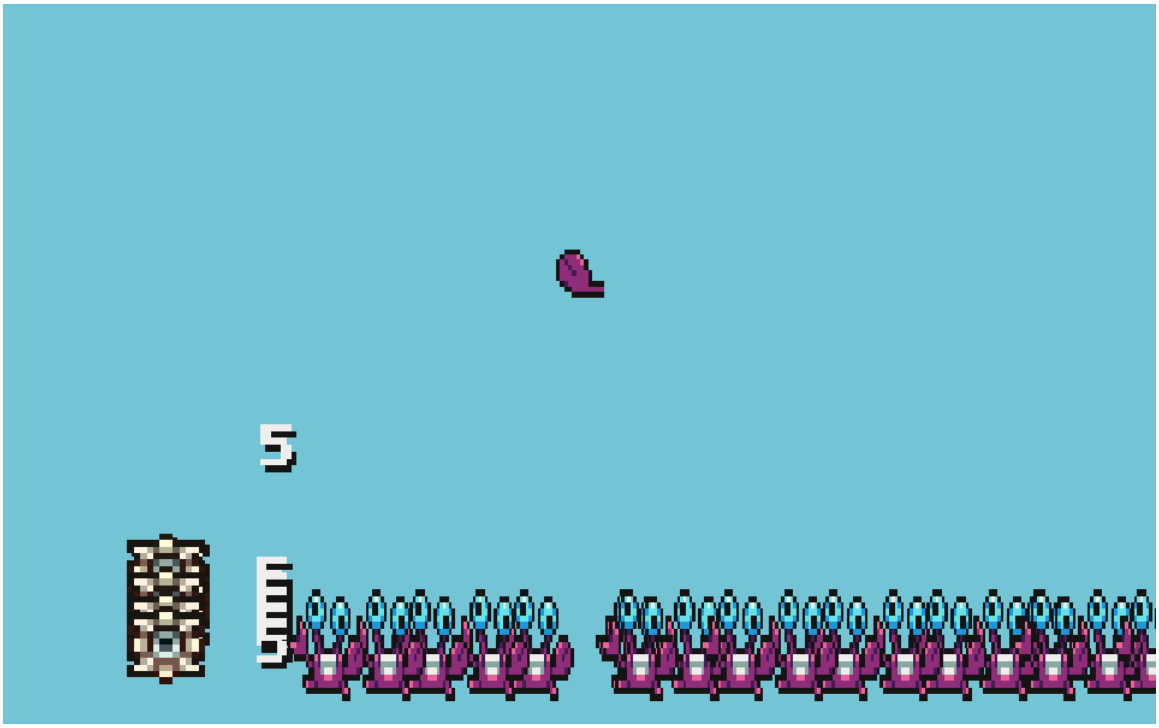

Figure 7.19: A screenshot taken from the beginning of the conceptual blend game.

http://guzdial.com/StudyGames/7/.

- **Compositional Adaptation (C):** The compositional adaptation approach lead to a game that I believe was the most like a standard platformer, given that it was essentially a smaller version of the amalgam game. The game again used the system's understanding of Kirby physics, but without the amalgamation requirement to incorporate as much information as possible it is more streamlined. The game is essentially a series of small, very simple platforming challenges. The only strange thing was the random patches of numbers floating in the air, having had decorative elements from other games mapped onto them. I include a screenshot of the start of the game in Figure 7.20. The game is playable at http://guzdial.com/StudyGames/8/.

7.5.6 Human Subject Study Method

The study was advertised through social media, in particular Twitter, Facebook, Discord, and Reddit. Participants were directed to a landing page where they were asked to review

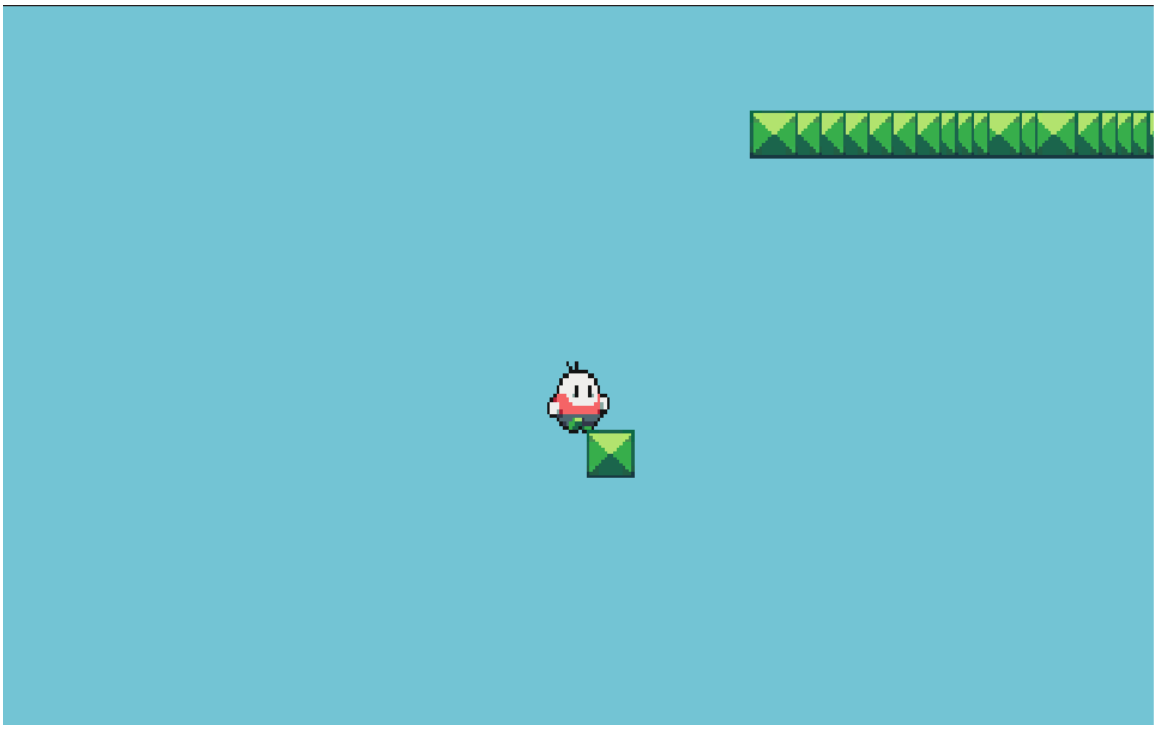

Figure 7.20: A screenshot taken from the beginning of the compositional adaptation game.

a consent form, and then press a button to retrieve an ID, the links to the three games they would play, and a link to the post-study survey. Each player played one human game, one expansion game, and one baseline game in a random order. The ID reflected this in its first three values and then had an additional 3 random digits. Players were asked to give each game " a few tries or until you feel you understand it". After this the player took part in the post-study survey.

The primary purpose of the survey was to determine how the expansion games compared to the baseline and human games in terms of novelty, surprise, and value. The participants were thus asked to rank the games across the same set of experiential features I applied in the level design study from Chapter 3. Specifically participants were asked to:

1. Please rank the three games according to how fun they were to play. The participants were given the options "Most Fun", "2nd Most Fun", and "Third Most Fun".

2. Please rank the three games according to how frustrating they were to play. The participants were given the options "Most Frustrating", "2nd Most Frustrating", and

"Third Most Frustrating".

3. Please rank the three games according to how challenging they were to play. The participants were given the options "Most Challenging", "2nd Most Challenging", and "Third Most Challenging".

4. Please rank the three games according to how surprising they were to play. The participants were given the options "Most Surprising", "2nd Most Surprising", and "Third Most Surprising".

5. Please rank the three games according to which of the games you would most like to play a complete, polished version of. The participants were given the options "Most Liked", "2nd Most Liked", and "Third Most Liked".

6. Please rank the three games according to which of the games you thought was the most creative. The participants were given the options "Most Creative", "2nd Most Creative", and "Third Most Creative".

Questions 1, 2, 3, and 5 were all meant to capture different elements of game value, though I note I focused on challenge and thus most focus on the answers to question 3. Question 4 spoke to how surprising the game was and question 6 to how creative the game was overall. I did not ask a question concerning novelty directly because novelty in this case was a measure of the difference between the output game and three specific games and I did not want to limit participants to only those who had played these games. Further, objectively I could measure that the conceptual expansion games had higher novelty metrics compared to the combinational creativity baseline games.

After each question participants had the option to leave a comment explaining their ranking with the text "(Optional) Why did you make that ranking?". Once the participant finished the ranking questions they were asked to assign which of the games they felt was made by a human and which by an AI with the text "For each of the three games you

played, which do you think was designed by a human and which by an AI?". Participants had been informed "some" of the games were made by humans and "some" by AI, but not which was which. I note this was made somewhat easier given that all the AI-generated games were made in Unity, while two of the three human games were made in javascript.

Following these questions reflecting on their experience participants were asked a series of demographic questions, most on a Likert scale. Specifically, participants were asked:

1. How often do you play video games generally? Participants were given the options daily, weekly, monthly, yearly, and never.

2. How often do you play platformer video games? Participants were given the same options as above.

3. How familiar are you with video game design/development? Participants were asked to give a rating between 1 (Novice) and 5 (Expert).

4. How familiar are you with Artificial Intelligence? Participants were asked to give a rating between 1 (Novice) and 5 (Expert).

5. What is your gender? Participants were given a textbox for input.

6. What is your age range? Participants were given the options 18-22, 23-33, 34-54, and 55+.

7.5.7 Human Subject Study Results

One-hundred and fifty-six participants took part in this online study. Of these, one-hundred and seventeen identified as male, twenty-seven as female, six as non-binary, and the remaining six did not answer the question. Eighty-eight of the participants placed themselves in the 23-33 age range, Forty-one in the 18-22 age range, and twenty-five in the 34-55 age range. There was a strong bias towards participants familiar with video game design and development, with one-hundred and eight participants ranking their familiarity

as 3 or more. Similarly, one-hundred and twelve of the participants played games at least weekly, though only eighty-two played platformer games at least monthly. These results suggest that this population should have been well-suited to evaluating video games. There was also a strong bias towards the participants being AI experts with eighty-one selecting four or more, with an additional thirty-two selecting three. Thus these participants should have been well-suited to evaluating AI-generated video games, though perhaps too skilled at differentiating between human and AI generated games.

My principle hypothesis was that the conceptual expansion games would do better in terms of value and surprise compared to the baseline games, with the human games considered a gold standard. Notably I do not include novelty in this as I have an explicit, objective measure for novelty, while both the value and surprise metrics depended on approximations of human perception. I can break these into the following hypotheses for the purpose of parsing the results.

H1. The expansion games will be more similar to the human games in terms of challenge in comparison to the baseline games.

H2. The expansion games will be overall the most surprising.

H3. The expansion games will be more like the human games in terms of the other attributes of game value.

H4. The expansion games will be more creative than the baseline games.

Hypothesis H1 essentially mirrors the measure I designed for value, creating games that are challenging in the ways that the existing, human-designed games are challenging. H2 in turn mirrors the measure I designed for surprise, which aims to be more surprising than the human-designed games. H3 is meant to determine the importance of the choice of heuristic when it comes to conceptual expansions. In other words, despite not designing a heuristic that reflects other types of game value like being fun, not frustrating, and leading to players

wanting to play more, does conceptual expansion still reflect these attributes of the original, human-designed games? H4 then is meant to determine if conceptual expansion on its own reflects the human creative process better than the existing combinational creativity approaches. In the following sections I break down different aspects of the results in terms of these hypotheses.

Notably participants had no issue differentiating between the human and artificially generated games outside of a few outliers. Further, the websites the games were played on differed between two of the human games and all other games due to not making use of Unity, which would make this task much easier. I therefore do not include these results in my analysis.

Quantitative Results By Author Type

The first step was to determine the aggregate results of people's experience with the games, broken into the three types I identified above (human, expansion, baseline). One can consider this as separating the games into different groups according to the nature of their source or author as long as one treats the baseline combinational creativity approaches as part of the same class of approach as in [183]. By bucketing the games into one of three author types there were essentially six possible conditions participants could be placed in, based upon the order in which they interacted with games from these three types. Because of randomly assigning participants, the one-hundred and fify-six participants were not evenly spread across these six conditions. The smallest number assigned was to the category in which participants played a human game first, an expansion game second, and a baseline game third, with only nineteen participants. Thus I randomly selected nineteen participants from each of the six categories, and use these one-hundred and fourteen participant results for analysis in this section.

I summarize the results according the the median value of each ranking in Table **??**. I report median values as the rankings are ordinal but not necessary numeric. I further

Table 7.3: A table comparing the median experiential ranking across the different games according to author type.

	Human	Expansion	Baseline
Fun	1st	3rd	2nd
Frustrating	2nd	2nd	2nd
Challenging	1st	2nd	3rd
Surprising	3rd	1st	2nd
Liked	1st	3rd	2nd
Creative	2nd	2nd	2nd

discuss the results more fully below and give the results of a number of statistical tests in terms of the four hypotheses. Notably due to the number of hypotheses and associated tests I use a significance threshold of $p < 0.01$. For all comparisons I run the paired Wilcoxon Mann Whitney U test, given that these are non-normal distributions.

Hypothesis H1 posits that the expansion games should rank more like the human challenge rankings than the baseline challenge rankings. The median values in Table **??** give some evidence to this. Further, there are significant differences between the human and baseline challenge rankings ($p = 9.62e - 06$), and between the expansion and baseline challenge rankings ($p = 7.61e - 09$). However, I am unable to reject that the human and expansion challenge rankings come from the same underlying distribution ($p = 0.041$). H1 is thus supported by these results.

H2 suggests that the expansion games should be more surprising than either of the other two types of games. The median ranking being first and therefore most surprising for the expansion games lends credence to this. Using the paired Wilcoxon Mann Whitney U test with appropriate alternatives, I found that the expansion games were ranked significantly higher in terms of surprise than both the human ($p = 1.95e - 13$) and baseline ($p = 4.83e - 06$) rankings. Further, the baseline games were ranked as more surprising than the human game rankings ($p = 3.81e - 09$). H2 is thus supported. These results further suggest that combinational creativity approaches may generally produce more surprising results than human generated games for this particular problem.

H3 suggested that the expansion games would be ranked more like the human games for other elements of game value, which I investigate in terms of the fun, frustrating, and most liked results. In terms of fun the median rankings do not support this hypothesis, with expansion coming third more frequently than the baseline ranking. However, the statistical test was unable to reject the null hypothesis that the baseline and expansion fun rankings came from the same distribution ($p = 0.39$). But the human rankings were significantly higher than both the baseline ($p = 1.03e{-}08$) and expansion rankings ($p = 1.23e{-}09$). For the frustrating question all of the games median rankings were exactly the same, all of them having a median rank of second. However, the statistical tests allow me to compare the distributions more evenly with the expansion games ranked as significantly more frustrating overall in comparison to both the human ($p = 1.61e - 03$) and baseline ($p = 1.84e - 04$) rankings. The human and baseline frustrating rankings were too similar to reject the that they arose from the same distribution ($p = 0.25$). Finally the most liked results parallel the fun results, with the human games ranked significantly higher than baseline ($p = 1.24e - 09$) or expansion ($p = 1.32e - 09$) rankings, but with the baseline and expansion rankings unable to reject the null hypothesis that they arose from the same distribution ($p = 0.66$). This evidence does not support H3, with the expansion games only found to be about as fun and liked as the baseline games, and more frustrating than these games.

H4 states that the expansion games will be viewed as more creative than the baseline games. However as with frustration the median ranking of all of these games suggests that the participants evaluated the games equivalently on this measure. Unlike frustration though the statistical tests are unable to pick apart these results, with no comparison able to reject that they derive from the same distribution ($p > 0.7$). Thus H4 is not supported by these results.

Quantitative Results By Game

There are several reasons why taking an aggregate view of the games may not be ideal. First, both in the case of the human and baseline games there are three very different authors, with the human games being created by three humans and the baseline games created by three distinct combinational creativity approaches. Second, the conceptual expansion games build off one another in terms of surprise and novelty, which cannot be captured in aggregate. Third, humans were ranking individual games, not the sources of these games.

For this section I instead analyze the results in terms of each game individually. Taken this way there can be thought to be 27 distinct categories in terms of a random selection of three of these nine games, and this is without taking into account the ordering of these games. Given the random assignment the results are too skewed to run statistical tests given these categories (there are only two participants who interacted with Kise's game (K), Expansion game 2 (E2), and the adaptation game (A), as an example). Thus instead I treat these rankings as ratings, given that a multi-way ANOVA cannot find any significant impact from ordering on any of the experiential factors ($p > 0.01$). In this case I determine the game with the fewest number of results (the blend with only forty-three participants), and randomly select this number of results for each of the nine games, for a total of three-hundred and eighty-seven ratings per experiential measure (compare this to the four-hundred and sixty eight total ratings available). I employ these ratings for the analysis in this section, due to the number of comparisons I employ the same conservative significant threshold of $p = 0.01$, and I use the unpaired Wilxocon Mann Whitney U test throughout this section.

The ratings give more nuanced results in terms of hypothesis H1. Hypothesis 1 states that the expansion games (E1, E2, and E3) should be more rated more like the human games (K, CD, and TP) than the baseline games (A, B, and C) in terms of challenge. For E2 and E3 there was no significant difference between their challenge ratings and the challenge ratings of any of the human games, but there were significant differences found between

Table 7.4: A table comparing the median ratings across each measure for each game. Median ratings of "1" marked in bold to make the table easier to parse.

	K	CD	TP	E1	E2	E3	A	B	C
Fun	**1**	**1**	**1**	2	3	3	2	2	2
Frustrating	2	2	3	2	**1**	**1**	2	2	2
Challenging	**1**	2	2	2	2	2	3	3	2
Surprising	3	3	3	2	**1**	**1**	2	2	2
Liked	**1**	**1**	**1**	2	3	2	2	3	2
Creative	2	2	2	2	2	2	2	2	2

the challenge ratings of the amalgamation (A) and blend (B) games. However, E2 and E3's challenge ratings did not differ significantly from the challenge ratings of the composition game (C). E1's challenge ratings only differed significantly with Kise's game (K), they were too like the other challenge ratings to differ significantly for any of the other games. Thus, while in aggregate the results support H1, taken individually there is only partial support for H1.

Hypothesis H2 suggests that the expansion games (E1, E2, and E3) should be rated more surprising than either the human games or baseline games. E2 and E3 both had significantly higher surprise ratings than all but the amalgamation (A) and blend (B) games. However, while not significant according to the unpaired Wilxocon Mann Whitney U test, their median surprise ratings were still higher than these two games, as demonstrated in Table **??**. Comparatively, E1 only had significantly higher surprise ratings than the three human games. There was no significant difference between E1's surprise ratings and the surprise ratings of any of the baseline games. This follows from the way the surprise portion of the heuristic functioned, which would have pushed E2 and E3 to have been considered more surprising than the expansion games output before them. Thus H2 is again supported.

Taken in aggregate the results in the prior section did not support H3, which stated that the expansion games would be evaluated more like the human games than the baseline games in terms of other measures of game value outside of challenge. Looking at the results by game the picture becomes more nuanced.

In terms of the fun ratings, while largely the games follow their aggregate rankings, E1 stands out as an outlier from the other two expansion games. In fact E1 is found to be rated significantly more fun compared to E2 ($p = 4.22e - 08$), E3 ($p = 3.601e - 05$), and the blend game ($p = 2.49e$). However, I cannot reject that this rating distribution comes from the same rating distribution as the amalgam ($p = 0.03$) and composition ($p = 0.27$) games. Similarly, there is no significant difference between E1 and CD ($p = 0.16$), both the remaining human games are rated significantly more fun ($p < 0.001$). This suggests that E1 is more fun than the blend game and roughly as fun as Chris DeLeon's game (CD), the amalgam game (A), and the composition game (C).

For the *frustration* ratings E2 and E3 stand out as clear outliers from the remaining games, being rated as the most frustrating the majority of the time. These games are clearly the cause of the expansion games being found to be significantly more frustrating when taken in aggregate compared to the other types of games, which individual tests confirm ($p < 0.01$). However, there is no significant difference comparing the E1 frustrating ratings and any of the other games. This suggests E1 is about as frustrating as all of the non-expansion games.

For the *liked* rating, which as a reminder related to the game the participant would most like to play a complete version, both E1 and E3's ratings differ from the aggregate expansion games median rating of 3. While E2 is significantly lower rated than all other games on this measure ($p < 0.01$), both E1 and E3 are not. E1 is rated significantly less in terms of this measure than Trenton Pegeas' Game (TP), and significantly higher rated than the blend game, with no significant difference from any of the other non-expansion liked ratings. E3 on the other hand is rated significantly lower than all three human games, with no other significant differences comparing any of the other liked rating distributions.

Overall these results suggest more support for H3, given that at least E1 seemed to be more like at least one of the human games in terms of measures of game value. This points to a potential issue with the heuristic, which potentially overly biased games to-

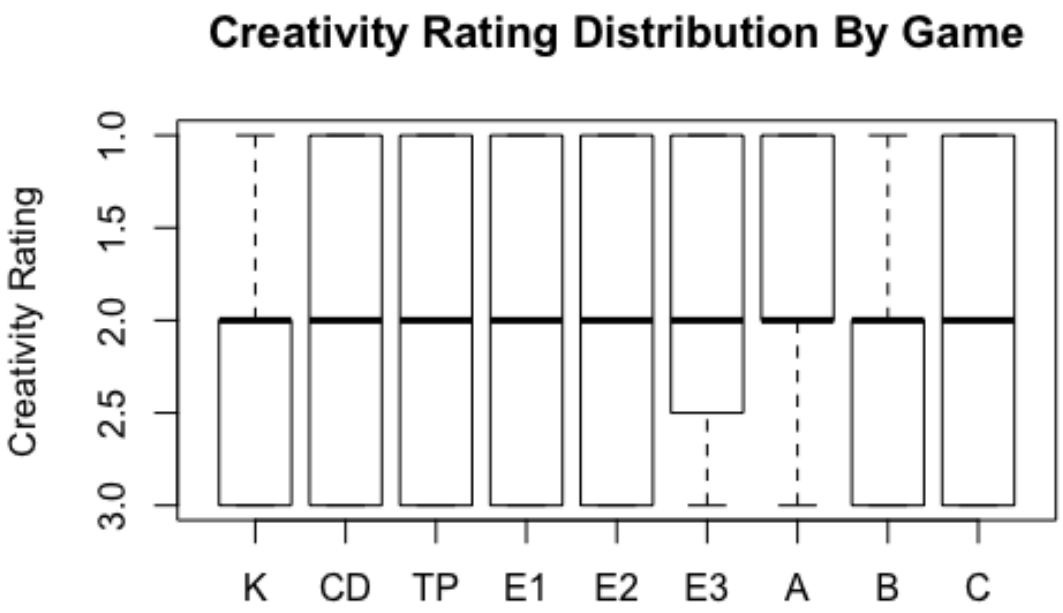

Figure 7.21: Boxplots of the creativity ratings for each of the games.

wards surprise and novelty at the cost of other positive elements of the input games that impacted these experiential features. The effect of this would have been compounded for E2 and compounded again for E3 given the changing knowledge bases associated with these games. I will discuss this further in the next section.

Given H4, I would expect to find the expansion games rated as more creative compared to the other games. However, breaking apart the games again lead to equal median values, pointing to the same issue identified in Chapter 3 that untrained participants appear to have difficulty consistently evaluating creativity. However, this is not the case as I visualize via boxplots in Figure 7.21. In particular, both E3 and the amalgam are rated significantly more creative than Kise's game ($p < 0.01$). However, it does not differ significantly from any of the other games ratings. Thus I can only consider there to be mixed support for H4.

7.5.8 Discussion

The results suggest strong support for H1 and H2, specifically that the games output by the conceptual expansion game generator closely relate to human made games in terms of challenge and are more surprising than human-made games or other combinational creativity games. Further, the output conceptual expansion games differed from the input games more than the output of historical combinational creativity approaches. This seems to follow from my choice of heuristic, which involved very specific measures for value, surprise, and novelty. Notably all the combinational creativity approaches used the same heuristic to search their spaces of potential output, but the output space of conceptual expansions seemed to have options that had higher measures of these values. This had both positive and negative consequences. While the expansion games outperformed the baselines in terms of challenge and all games in terms of surprise, the latter two games seemed to gain surprise at the expense of other types of game value besides challenge (specifically being less fun, more frustrating, and less liked). This could in part be due to the nature of the heuristic, where novelty and surprise were the majority of the heuristic value or due to the heuristic only focusing on challenge as a representation of value.

There are two major limitations of this game generation process and evaluation. First, I focused entirely on the genre of platformers instead of a more general genre-agnostic game generation approach. Second, the difficulty of evaluating the open ended creative expression of video game creation.

There are a great number of game genres outside of platformers. Some game genres do lend themselves better to the game graph representation and my machine learning approaches for level design and game rules. For example, it would be easy to extend this to two-dimensional action adventure games or racing games where all action occurs on the screen (e.g. each frame of the game can be considered a Markovian or semi-Markovian state) and where spatial relationships are of primary importance. However, in genres where this is not the case this game generation process would struggle. For example, in games

with hidden information like certain puzzle games, interactive fiction game, or any games that substantially rely on text.

Focusing on platformers for this work was helpful in terms of facilitating evaluations and opening up the possibility of future work. There has been prior work in platformer level design, which I could then draw upon for the sake of comparison to validate the quality of the level design information in the game graphs, as I did in Chapter 3. Further, platformers and the Mario series in particular represents the most popular video games financially [184]. I could therefore reasonably expect sufficient interest and expertise for the human subject study of the third variation. In terms of future works, platformers can be understood as a simplification of reality with simplified structure and physics. Thus it made sense that if I could demonstrate success in this genre it should be able to transfer to other genres. However, I recognize I have not yet done that work.

Evaluating video game creation or any open-ended creative task is a challenge. Thus I focused on an objective evaluation for variation 1 and did not attempt a formal evaluation of variation 2. However, because of this there are some potential issues for variation 3. In my attempt to make the comparison task easier by having all of the games use the same spritesheet and work under similar constraints there is the possibility that I biased the human game authors. For example, take the survey question asking participants to rank the games in terms of surprise. Given that the human game authors were requested to make a platformer "interpreted however [the author] liked" and the artificially authored games did not generally resemble platformers this may have lead participants to rank the artificial games more highly in terms of surprise. However, I note two pieces of evidence First, the human authors did not all interpret platformer the same way, in particular Chris DeLeon, who created more of a puzzle game. Second, the baseline games also did not strongly resemble platformers, but the expansion games were still ranked as more surprising than those overall.

I could have phrased the platformer requirement to the human authors in a way that

specifically paralleled how the artificial approaches worked. Ideally this would then alleviate the potential issue of biasing the authors too strongly towards standard platformer game designs. For example, I could have asked the human authors to make a game inspired by ”Mario, Mega Man, and Kirby”. However I would anticipate this having the opposite effect, too closely constraining the authors to create games like those three games. As an example I would not expect to see a game like Chris DeLeon's puzzle platformer game with those instructions. None of the three input games had single-screen puzzles like Chris' game that required specific sequences of actions to solve.

7.6 Conclusions

In this chapter I presented three variations of conceptual expansion over game graphs for game generation. Taken together this represents my attempted solution to the video game invention problem. As stated in the first chapter, the video game invention problem would require that the given knowledge base was insufficient to create a novel, surprising, and valuable answer given non-computational creativity approaches. I demonstrated this most clearly in the first variation's evaluation, where both the GA and conceptual blending approaches, given the same knowledge base, were outperformed by conceptual expansion. Thus, the question is whether the output games represented a novel, surprising, and valuable answer. The answer to the first one is a clear yes, given objective measures of novelty and given the human-validated measure of surprise. The last is less clear, but it is certainly true that for particular measures of value in this domain conceptual expansion produced valuable output.

These results indicate the importance of heuristics or other types of content selection strategies when it comes to conceptual expansion. The heuristics in this chapter were focused on game challenge and biased towards novelty and surprise. This was partially due to the fact that there exist no validated, computational measures of general value in the domain of games. Conceptual expansion still outperformed its baselines in terms of the particular

heuristic I employed (thus creating games with challenge like a human-designed games when I rewarded that). However, conceptual expansion search did not replicate other valuable elements of its input games outside of challenge. This suggests that in domains with more clear computational measures of value, conceptual expansion search may do well as it would be incentivized to retain or create structure that reflected that value. Further, a potential criticism of conceptual expansion thus far may be that its application is limited to games. In this next chapter I investigate both a more clear heuristic and the novel domains of image classification and image generation.

CHAPTER 8

COMBINETS: APPLYING CONCEPTUAL EXPANSION TO NEURAL NETWORKS

Modern deep learning systems perform well with large amounts of training data on known classes but often struggle otherwise. This is a general issue given the invention or discovery of novel classes, rare or illusive classes, or the imagining of fantastical classes. For example, if a new traffic sign were invented tomorrow it would have a severe, negative impact on autonomous driving efforts until enough training examples were collected.

Deep learning success has depended more on the size of datasets than on the strength of algorithms [2]. A significant amount of training data for many classes exists. But there are also many novel, rare, or fantastical classes with insufficient data that can be understood as derivations or combinations of existing classes. For example, consider a pegasus, a fantastical creature that appears to be a horse with wings, and therefore can be thought of as a combination of a horse and a bird. If we suddenly discovered a pegasus and only had a few pictures, we couldn't train a typical neural network classifier to recognize a pegasus as a new class nor a generative adversarial network to create new pegasus images. However we might be able to approximate both models given existing models trained on horse and bird data via conceptual expansion.

In this chapter I investigate the application of conceptual expansion to deep neural network models. This can be thought of as equivalent to the invention problem for image classification and generation neural networks. This chapter allows me to demonstrate the potential of conceptual expansion in domains with strong selection criteria (e.g. accuracy for image classification). It also allows me to demonstrate the application of conceptual expansion to more general data structures (neural networks) outside of the novel representations I have developed throughout the rest of these chapters in my approach to the video

game invention problem. Finally, this problem domain allows me to demonstrate the application of conceptual expansion to problems with a single input model, which is made possible by conceptual expansions component-based combination function.

The remainder of this chapter is organized as follows. I briefly cover related background work, specifying those existing works that are relevant to the invention problems for image classification and generation. I then discuss the application of conceptual expansion to this domain and my alteration of the conceptual expansion search algorithm presented in the prior chapter. I then present a series of experiments for image classification neural networks and image generation neural networks to clarify the limitations and strengths of conceptual expansion in this domain.

8.1 Background

Various approaches exist that reuse knowledge from models trained on large datasets for a particular problem to try to solve problems with smaller datasets, such as zero-shot and transfer learning. In these approaches, knowledge from a trained source model is applied to a target problem by either retraining the network on the target dataset [133] or leveraging sufficiently general or authored features to represent new classes [126]. The latter of these two approaches is not guaranteed to perform well depending on source and target problems, and the former of these is limited in terms of what final target models can be learned. Many implementations of these approaches rely on some secondary feature representation that is either hand-authored [129, 131, 132] or learned from a secondary dataset. [133, 134, 135, 136, 137]. As an alternative one can author an explicit model of transfer such as metaphors [133] or hypotheses [140]. In this chapter I focus only on the prior work that does not include any secondary knowledge representation, either human-authored or learned from some other dataset given that I focus specifically on the problem of recombining existing knowledge to create new knowledge.

8.2 Conceptual Expansion in Neural Networks

Imagine tomorrow we discover that a pegasus exists. Initially we would lack enough images of this newly discovered flying horse to build a traditional classifier or image generator. However, suppose we have neural network classifiers and generators trained on classes including horses and birds. Conceptual expansion could allows us to reuse the learned features from machine learned model(s) to produce new models without additional training or additional transfer features.

One can consider the case where a class or concept ($class_X$) is a combination of other classes ($class_1, ...class_n$) and that the learned features of models of classes $class_1, ...class_n$ can be recombined to create the features of a model of $class_X$. In these cases, I hypothesize that conceptual expansions can represent models one cannot necessarily discover using conventional machine learning techniques with the available data. Furthermore, I hypothesize that these conceptual expansion models may perform better on specific tasks than standard models in cases with small amounts of available data, such as identifying or generating new classes of objects.

One can use a heuristic informed by this small amount of training data to guide the selection for the final conceptual expansion. As a reminder a conceptual expansion of concept X is represented as the following function:

$$\mathrm{CE}^{\mathrm{X}}(F, A) = a_1 * f_1 + a_2 * f_2 ... a_n * f_n \tag{8.1}$$

Where $F = \{f_1, ...f_n\}$ is the set of all mapped features and $A = \{a_1, ...a_n\}$ is a set of filters each dictating what of and what amount of mapped feature f_i should be represented in the final conceptual expansion. In the ideal case $X = CE^X$ (e.g. a combined model of birds and horses equals the ideal pegasus model). The exact shape of a_i depends upon the feature representation. If features are matrices, as in a neural net, each a_i is also a matrix. In the case of matrices the multiplication is an element-wise multiplication or

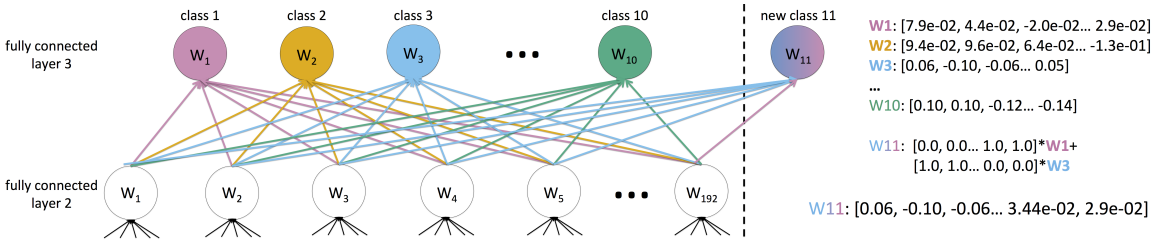

Figure 8.1: Visualization of the initial construction of the new (eleventh) class based on the mapping.

Hadamard product. As an example, in the case of neural image recognition, $\{f_1, ..., f_n\}$ are the variables in a convolutional neural network learned via backpropagation. Deriving a conceptual expansion is the process of finding an A for known features F such that $\mathrm{CE}^{\mathrm{X}}(\cdot)$ optimizes a given objective or heuristic towards some target concept X.

In this representation, the space of conceptual expansions is a multidimensional, parameterized search space over possible combinations of any input models. There exists an infinite number of possible conceptual expansions for non-symbolic features, which makes naively deriving this representation ill-advised. As in the prior chapter the approach requires one to first derive a *mapping*. The mapping determines what particular prior knowledge—in this case the weights and biases of a neural network—will be combined to address the novel case. This will determine the starting point of the later search process for exploring the space of possible conceptual expansions.

Given a mapping, one can construct an initial conceptual expansion—a set of $F = \{f_1, ..., f_n\}$ and an $A = \{a_1, ..., a_n\}$—that is iterated upon to optimize for domain specific notions of quality (in the example pegasus scenario image recognition accuracy). In the following sections I discuss the creation of the mapping and then the refinement of the conceptual expansion.

8.2.1 Mapping Construction

Constructing the initial mapping is relatively similar to the mapping construction in the previous chapter. As input I assume there exists an existing trained model or models (CifarNet trained on CIFAR-10 for the purposes of this example [185]), and data for a novel class

Algorithm 4: Conceptual Expansion Search

input : available data $data$, an initial model $model$, a mapping m, and a score $score$
output: The maximum expansion found according to the heuristic

```
1  maxE ← DefaultExpansion(model)+m;
2  maxScore ← score;
3  v ← [maxE ];
4  improving ← 0;
5  while improving < 10 do
6      n ← maxE.GetNeighbor(v);
7      v ← v + n;
8      s ← Heuristic(n, data);
9      oldMax ← maxScore;
10     maxScore, maxE ← max([maxScore, maxE ], [s, n ]);
11     if oldMax < maxScore then
12         improving ←improving +1;
13     else
14         improving ←0;
15 return maxE;
```

(whatever pegasus images we have). I construct a mapping with the novel class data by examining how the model or models in the knowledge base perform on the data for the novel class. The mapping is constructed according to the ratio of the new images classified into each of the old classes. For example, suppose one has a CifarNet trained on CIFAR-10 and additionally one has four pegasus images. Say CifarNet classifies two of the four pegasus images as a horse and two as a bird. In that case construct a mapping of: f_1 consisting of the weights and biases associated with the horse class, and f_2 consisting of the weights and biases associated with the bird class. The A values for both variables would be be 0.5—the classification ratio—meaning a floating point a value for the biases and an a matrix for the weights. I visualize this process in Figure 8.1.

8.2.2 Conceptual Expansion Search

The space of potential conceptual expansions grows exponentially with the number of input features, and the mapping construction stage gives an initial starting point in this space from which to search. I present the pseudocode for this chapter's variation on the conceptual expansion search in Algorithm 4. Line 1 creates an initial expansion by combining a default

expansion with the mapping information. The exact nature of this depends on the final network architecture. For example, the mapping may overwrite the entirety of the network if the input models and final model have the same architecture or just the final classification layer if not (as in the case of adding an additional class). In this case a default expansion is a conceptual expansion equivalent to the original model(s), in that each variable is replaced by an expanded variable with its original f_i value and an a_i of 1.0 (or matrix of 1.0's). This means that the initial expansion is functionally identical to the original model, beyond any weights impacted by the mapping. This initial conceptual expansion derived at the end of the mapping construction will be a linear combination of the existing knowledge, but the final conceptual expansion need not be a linear combination.

Once the process has a mapping this process searches for a set of F and A for which the conceptual expansion performs well on a domain-specific measure $Heuristic$ (e.g. pegasus classification accuracy). As with the prior chapter I employ a greedy optimization search that checks a fixed number of neighbors before the search ends. The $GetNeighbor$ function randomly selects between one of the following: altering a single element of a single a_i, replacing all of the values of a single a_i replacing values of x_i with a randomly selected alternative x_j, or adding an additional x_i and corresponding random a_i to an expanded variable. Notably in this algorithm I include a set of visited expansions v in order to avoid sampling a neighbor that has already been visited. The final output of this process is the maximum scoring conceptual expansion found during the search. For the purposes of clarity I refer to these conceptual expansions of neural networks as *combinets*.

8.3 CifarNet Experiments

In this section I present a series of experiments meant to demonstrate the strengths and limitations of conceptual expansions for image classification with deep neural networks. I chose CIFAR-10 and CIFAR-100 [185] for these experiments as they represent well-understood datasets. It is not my goal to achieve state of the art on CIFAR-10 or CIFAR-

Table 8.1: A table with the average test accuracy for the first experiment. The orig. column displays the accuracy for the 10,000 test images for the original 10 classes of CIFAR-10. The 11th column displays the accuracy for the CIFAR-100 test images.

	100		50		10		5		1	
Fox	11th	orig.	11th	orig.	11th	orig.	11th	orig.	11th	orig.
combinet	**34.0±3.5**	81.8±2.2	**26.0±5.2**	81.59±1.9	**28.3±3.5**	79.1±1.6	**23.0±8.5**	80.6±1.2	**12.0±9.8**	80.7±7.2
standard	7.0±2.7	62.04	0.0±0.0	62.17	0.0±0.0	62.34	0.0±0.0	62.44	0.0±0.0	76.44±3.5
transfer	5.0±4.3	**87.2±0.5**	0.0±0.0	**87.9±0.2**	0.0±0.0	**88.1±0.4**	0.0±0.0	**87.7±0.2**	0.0±0.0	**88.0±1.1**
zero-shot	11.0±0.7	86.2±0.4	11.0±1.0	86.2±0.8	9.6±2.3	86.2±0.2	10.0±4.6	86.0±1.4	6.0±3.3	83.2±2.5
Plain	11th	orig.	11th	orig.	11th	orig.	11th	orig.	11th	orig.
combinet	**53.0±10.0**	84.0±3.6	**45.7±7.6**	84.2±7.8	**31.3±22.0**	83.9±2.4	**28.3±12.6**	82.3±2.2	**23.0±17.4**	84.0±2.4
standard	50.0±7.7	62.54	42.0±3.2	62.18	16.0±12.8	61.67	0.0±0.0	62.27	0.0±0.0	62.27
transfer	4.5±3.0	**86.92**	0.0±0.0	**86.91**	0.0±0.0	**86.96**	0.0±0.0	**87.20**	0.0±0.0	**87.20**
zero-shot	23.0±0.7	86.2±0.5	23.6±1.1	86.2±0.3	22±2.8	86.1±13.9	18.6±3.8	83.7±3.4	15.6±7.3	82.7±2.9

100; I instead use these datasets to construct problems in which a system must identify images of a class not present in some initial training set given limited training data on the novel class. I then apply my approach to these problems, comparing them with appropriate baselines. For the source deep neural network model I chose CifarNet [185], again due to existing understanding of its performance on the more traditional applications of these datasets. I chose not to make use of a larger dataset like ImageNet or a larger architecture [186], as I aim to compare how conceptual expansion can construct new features given a limited set of input features. I do not include a full description of CifarNet but note that it is a two-layer convolutional neural net with three fully-connected layers.

For each experiment, I ran the conceptual expansion search algorithm ten times and took the most successful combinet found across the ten runs in terms of training accuracy. I did this to ensure I had found a near optimal conceptual expansion. This combined with the inclusion of a visited set improves the optimality of the conceptual expansion search algorithm compared to the prior chapter. However, I still anticipate that future work will explore more sophisticated optimization strategies. Further, I note that this approach was still many times faster than initially training the CifarNet on CIFAR-10 with backpropagation.

The first experiment expands a CifarNet trained on CIFAR-10 to recognize one additional class selected from CIFAR-100 that is not in CIFAR-10. I vary the amount of training data available for the newly introduced class. This allows me to evaluate the per-

formance of the conceptual expansion approach under a variety of controlled conditions. The second experiment fully expands a CifarNet model trained on CIFAR-10 to recognize the one-hundred classes of CIFAR-100 with limited training data. Finally, I investigate the running example throughout this paper: expanding a CifarNet model trained on CIFAR-10 to classify pegasus images.

8.3.1 CIFAR-10 + Fox/Plain

For the initial experiment I chose to add fox and plain (as in a grassy field) recognition to the CifarNet, as these classes exist within CIFAR-100, but not within CIFAR-10. It is worth noting that CIFAR-10 is made up of the classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. I chose foxes and plains for this initial case study because they represented illustrative examples of conceptual expansion performance. There exists a number of classes in CIFAR-10 that can be argued to be similar to foxes, but no classes similar to plains. Thus in the former case learning features capable of discriminating between fox and the similar classes in CIFAR-10 will be important.

For training data I drew from the 50,000 training examples for the ten classes of CIFAR-10, adding a varying number of training instance of fox or plain. For test data I made use of the full 10,000 CIFAR-10 test set and the 100 samples in the CIFAR-100 test set for each class. For each size slice of training data (i.e. 1, 5, 10, 50, and 100) I constructed five unique random slices. I chose five for consistency across all the differently sized slices, given that there was a maximum of 500 training images for fox and plain, and the largest slice size was 100. I present the average test accuracy across all approaches and with all sample sizes in Table 8.1. This table shows the results when I provide five slices of fox or plain images in the quantities of 1, 5, 10, 50, or 100. For each slice, I provide the accuracy on the original CIFAR-10 images and the accuracy of identifying the 11th class (either fox or plains).

I evaluate against three baselines. The first baseline (standard) trains CifarNet with

backpropagation with stratified branches on the 10,000 CIFAR-10 images and newly introduced foxes or plains. This baseline makes the assumption that the new class was part of the same domain as the other classes as in [128]. For the second baseline I took inspiration from transfer learning and student-teacher models [138, 139, 187], and train an initial CifarNet on only the CIFAR-10 data and then retrain the classification layers to predict the eleventh class with the newly available data. I note that transfer learning typically involves training on a larger dataset, such as ImageNet, then retraining the final classification layer. However, I wished to compare how these different approaches alter the same initial features for classifying the new class. For the third baseline I drew on the zero-shot approach outlined in [146], using the average activation of the trained CifarNet classification layer on the training data to derive feature classification vectors. In all cases I trained the model until convergence.

There exist many other transfer approaches, but other approaches tend to require additional human authoring of transfer methods or features and/or an additional dataset to draw from. I focus on comparing the behavior of these approaches in terms of altering or leveraging learned features. I did not draw on any historical combinational creativity approaches for baselines given that in this domain I cannot treat the components as bags of symbolic features as I do in all prior chapters.

As can be seen in Table 8.1, the combinet consistently outperforms the baselines at recognizing the newly added eleventh class. I note that the expected CifarNet test accuracy for CIFAR-10 is 85%. Combinets achieve the best accuracy on the newly added class while only losing a small amount of accuracy on average on the 10 original classes. The combinet loss in CIFAR-10 accuracy was almost always due to overgeneralizing. The transfer approach did slightly better than the expected CIFAR-10 accuracy, but this matches previously reported accuracy improvements from retraining [187].

Foxes clearly confused the baselines, leading to no correctly identified test foxes for the standard or transfer baselines at the lowest values. Compared to plains, foxes had sig-

nificant overlap in terms of features with cats and dogs. With these smaller size samples transfer and standard were unable to learn or adapt suitable discriminatory features. Comparatively, the conceptual expansion approach was capable of combining existing features into new features that were more successfully able to discriminate between these classes. The zero-shot approach did not require additional training and instead made use of secondary features to make predictions, which was more consistent, but still not as successful as my approach in classifying the new class. In comparison plain was much easier to recognize for the baselines, likely due to the fact that it represented a class that differed significantly from the existing ten. However, combinets were still able to outperform the baselines, with novel features that could better differentiate the plain class.

Note that combinets do not always outperform these other approaches. For example, the standard approach beats out combinets, getting an average of 83% accuracy with access to all 500 plain training images, while the combinet only achieves an accuracy of roughly 50%. This suggests that combinets are only suited to problems with low training data with this current approach.

8.3.2 Expanding CIFAR-10 to CIFAR-100

For the prior experiments I added a single eleventh class from CIFAR-100 to a CifarNet trained on CIFAR-10. This experiment looks at the problem of expanding a trained CifarNet from classifying the ten classes of the CIFAR-10 dataset to the one-hundred classes of the CIFAR-100 dataset.

For this experiment I limited the training data to ten randomly chosen samples of each CIFAR-100 class. I altered the approach to account for the change in task, constructing an initial mapping for each class individually as if we were expanding a CifarNet to just that eleventh class. I utilized the same three baselines as with the first experiment.

I note that one would not typically utilize CifarNet for this task. Even given access to all 50,000 training samples of CIFAR-100 a CifarNet trained using backpropagation only

Figure 8.2: The fifteen pegasus images found via Flickr and resized to the standard CIFAR-10 dimensions of 32x32 pixels.

achieves roughly 30% test accuracy for CIFAR-100. I mean to show the relative scale of accuracy with conceptual expansion and not attempt to achieve state of the art on CIFAR-100 with the full dataset. I tested on the 100,000 test samples available for CIFAR-100.

The average test accuracy across all 100 classes are as follows: the combinet achieves 11.13%, the standard baseline achieves 1.20%, the transfer baseline achieves 6.43%, and the zero-shot baseline achieves 4.10%. I note that this approach is the only one to do better than chance, and significantly outperforms all other baselines. However no approach reaches anywhere near the 30% accuracy that could be achieved with full training data for this architecture.

8.3.3 Pegasus

I return to the running example of an image recognition system that can recognize a pegasus. Unfortunately I lack actual images of a pegasus. To approximate this I collected fifteen photo-realistic, open-use pegasus images from Flickr, these can be found in Figure

8.2. Using the same combinet as the above two experiments I ran a 10-5 training/test split and a 5-10 training/test split. In this case the training data was the first 10 and first 5 images from Figure 8.2 respectively. For the former the final model recognized 4 of the 5 pegasus images (80% accuracy), with 80% CIFAR-10 accuracy, and for the latter it recognized 5 of the 10 pegasus images (50% accuracy) with 82% CIFAR-10 accuracy.

8.4 DCGAN Experiment

In this section I demonstrate the application of conceptual expansions to generative adversarial networks (GANs). Specifically, I demonstrate the ability to use conceptual expansions to find GANs that can generate images of a class without traditional training on images of that class. I also demonstrate how this approach can take as input an arbitrary number of initial neural networks, instead of the one network for the classification experiments. This more closely parallels taking three input game graphs from the prior chapter. I make use of the DCGAN [188] as the GAN architecture for this experiment, as it has known performance on a number of tasks. I make use of the CIFAR-100 dataset from the prior section and in addition use the Caltech-UCSD Birds-200-2011 [189], the CAT [190], the Stanford Dogs [191], FGVC Aircraft [192], and the Standford Cars [193] datasets. I make use of these five datasets as they represent five of the ten CIFAR-10 classes, but with significantly more images and images of higher quality. Sharing classes between experiments allows one to draw comparisons between results.

I trained a DCGAN on each of these datasets till convergence, then used all five of these models as the original knowledge base for this approach. Specifically, I built mappings by testing the proportion of training samples the discriminator of each GAN classified as real. I then built a combinet discriminator for the target class from the discriminators of each GAN. This is equivalent to the process used for deriving Cifarnet image classifiers above, with the discriminator treated as a two-class classifier (real or fake). Finally I built a combinet generator from the generators of each GAN, using the combinet discriminators as the

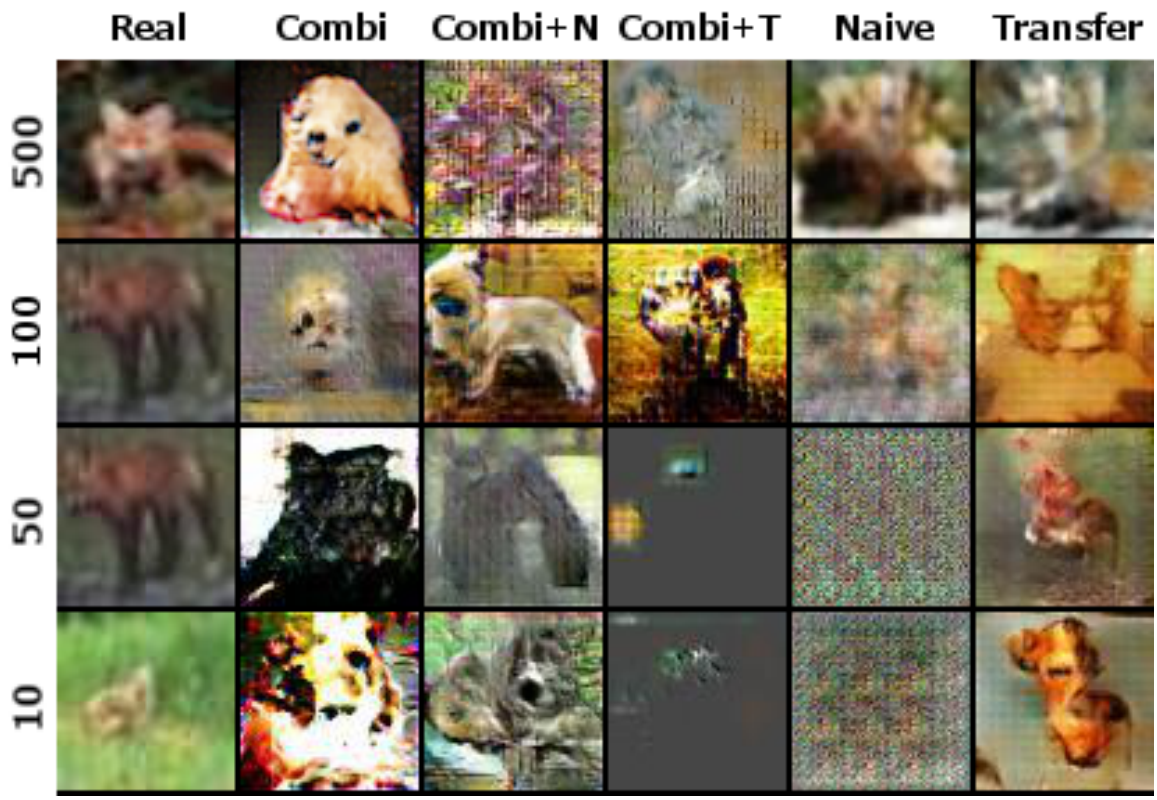

Figure 8.3: Most fox-like output according to the model for each baseline and sample size.

heuristic for the conceptual expansion search. I nickname these combinet discriminators and generators *combiGANs*. As above I made use of the fox images of CIFAR-100 for the novel class, varying the number of available images.

I built two baselines: (1) A naive baseline, which involved training the DCGAN on the available fox images in the traditional manner. (2) A transfer baseline, in which I took a DCGAN trained on the Stanford Dogs dataset and retrained it on the fox dataset. I also built two variations of combiGAN: (1) A combiGAN baseline in which I used the discriminator of the naive baseline as the heuristic for the combinet generator (Combi+N). (2) Same as the last, but using the transfer baseline discriminator (Combi+T). I further built a baseline trained on the Stanford Dogs, CAT dataset, and Fox images simultaneously as in [143], but found that it did not have any improvement over the other baselines. I do not include the zero shot baseline from the prior section as it is only suitable for classification tasks.

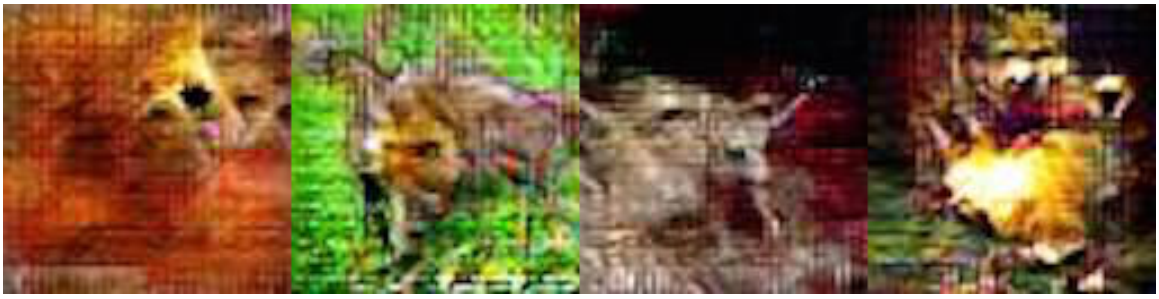

Figure 8.4: Four fox-like images hand-picked by the authors from the first 1,000 images output by the combiGAN trained on 500 foxes.

Table 8.2: Summary of results for the GAN experiments.

	combiGAN		combi+N		combi+T		Naive		Transfer	
Samples	I	KL	I	KL	I	KL	I	KL	I	KL
500	3.83±0.32	0.33	**4.61±0.22**	**0.28**	3.05±0.23	0.31	2.98±0.25	0.33	3.38±0.19	1.05
100	4.23±0.15	**0.10**	4.38±0.37	0.29	**4.40±0.19**	0.43	1.76±0.04	0.33	3.26±0.23	0.36
50	**4.05±0.24**	0.22	4.03±0.35	**0.12**	1.69±0.05	2.36	1.06±0.00	10.8	3.97±0.22	0.21
10	4.67±0.44	0.44	**4.79±0.28**	0.13	3.06±0.19	1.20	1.20±0.01	10.8	4.40±0.19	**0.11**

8.4.1 CombiGAN Results

I made use of two metrics: the inception score [194] and Kullback-Leibler (KL) divergence between generated image classification and true image classification distributions. I acknowledge that inception score was originally designed for ImageNet; since I do not train on ImageNet, I cannot use this as an objective score, but I can use it as a comparative metric of objectness. For the second metric I desired some way to represent how fox-like the generated images were. Thus I made use of the standard classifier trained on 500 foxes, though I could have made use of any classifier in theory. I compare the distribution over classes of real CIFAR-100 fox images and the fake images with the KL divergence. I generated 10,000 images from each GAN to test each metric. I summarize the results of this experiment in Table 8.2.

I note that in almost all cases my approach or one of its variations (combi+N and combi+T) outperform the two baselines. In the case with 10 training images the transfer baseline beats my approach on the fox-like measure, but this 0.11 differs only slightly from the 0.13 combi+N value. In Figure 8.3, I include the most fox-like image in terms

of classifier confidence from the training samples (real) and each baseline's output. I note that the combiGAN output had a tendency to retain face-like features, while the transfer baseline tended to revert to fuzzy blobs. I also include four hand-picked fox-like images from the 500 sample case in Figure 8.4.

8.5 Discussion and Limitations

Conceptual expansion applied to neural networks—combinets and combiGANs—outperforms standard approaches on image classification and generation problems with limited data and without additional knowledge engineering.

I anticipate the future performance of conceptual expansion of deep neural networks to depend upon the extent to which the existing knowledge base contains relevant information to the new problem and the existence of a domain-appropriate content selection function (e.g. training accuracy for image classification). In the last chapter I demonstrated the behavior of conceptual expansion with a heuristic that did not fully reflect value in that domain. Comparing the results to the image classification domain, where accuracy is a clear representation of value, the importance of a heuristic that reflects this domain-specific value is clear. This can be further seen given that the conceptual expansion approach performed more successfully with image classification compared to image generation, though it outperformed appropriate baselines in both cases.

Conceptual expansion of deep neural networks appears less dependent on training data than existing transfer learning approaches as evidenced by the comparative performance of the approach with low training data, This is further evidenced by those instances where conceptual expansion outperformed itself with less training data. I anticipate further exploration of this in future work, investigating what important knowledge base features lead to successful conceptual expansion applications.

I expect these results to generalize to other image classification problems, but recognize my choice of datasets as a potential limiting factor. CIFAR-10 and CIFAR-100 have very

low resolution images (32x32 RGB images). Further, I do not make use of traditional data augmentation techniques such as noising or horizontal flips of the images. I note once again that I chose these datasets for these experiments to focus on feature adaptation. My purpose was never to achieve state-of-the-art performance.

CHAPTER 9
CONCLUSIONS

In this final chapter I reflect on my thesis statement, the work I presented in this document, and several potential avenues for future work. At the beginning of this document I gave this statement:

> For creativity problems that require the combination of aspects of distinct examples, conceptual expansion of generative or evaluative models can create a greater range of artifacts or behaviors, with greater measures of value, surprise, and novelty than standard combinational approaches or approaches that do not explicitly model combination.

I presented evidence towards this statement in two primary domains: game design and image classification, with secondary domains of game level design, game level evaluation, and image generation. I demonstrated the greater range of artifacts in Chapter 6, in a comparative study of the relative output spaces of conceptual expansion, amalgamation, conceptual blending, compositional adaptation, and random combinations. I demonstrated the greater measures of value in the hold-one-out quantitative game graph study (Chapter 7), the combinets quantitative study (Chapter 8), and the human subject study (Chapter 7). Notably these studies demonstrate the importance of the phrase "measures of" in this statement. As the results of the human subject study demonstrated, how these measures are defined directly impacts the performance of the approach.

It may be unclear from this statement when it is appropriate to apply conceptual expansion to a novel problem domain. There are a number of costs in the application of conceptual expansion. First, there must be some knowledge base of existing solutions to a related problem domain. For example, the trained game graph representations discussed in

Chapter 7 or the deep neural network models (DNNs) in Chapter 8. There is a development time cost (as with the game graphs) in instances where no existing knowledge representation exists and regardless there is a training or encoding task time cost for filling in these knowledge representations. Second, the instances in this knowledge base must be in a modular knowledge representation, having individual components that can be recombined (such as the nodes of the game graph or the weights of the DNNs). This restricts the type of problems one can apply conceptual expansion as there are knowledge representations that are more or less modular. For example, it is much easier to break apart and recombine graphs or networks than images or strings. Third, there must be some initial way to establish a mapping, which determines an initial starting conceptual expansion and gives a good basis for how to apply the knowledge from the knowledge base to solve a new problem. There is a development cost here for any new problem domain as it requires domain-specific and representation-specific knowledge.

The process described above can then allow me to more clearly state the claims implicit in the thesis statement. For problem domains in which it is possible to undertake the above process and which it is possible to learn or encode sufficient existing knowledge that a recombination of this knowledge can create a solution, conceptual expansion can outperform existing combinational creativity approaches and domain-specific baselines given the same knowledge. The issue then becomes how one determines when one has sufficient existing knowledge to solve some problem. In some cases, as in adding fox recognition to an image classifier capable of recognizing cats and dogs, there is an intuitive understanding of when there is sufficient knowledge (cats and dogs). I note that I have not yet derived domain-agnostic metrics or qualities to determine when this is the case more generally. I leave this for future work.

For the remainder of this chapter I discuss my contributions in terms of major areas of impact and potential future work.

9.1 Contributions

I frame my discussion of contributions in terms of two areas that I view as benefiting the most from this work: procedural content generation via machine learning (PCGML) [15] and computational creativity.

9.1.1 PCGML

Procedural content generation via machine learning (PCGML) is the study of automatically generating game content via models trained on existing game content. The area of primary focus for PCGML thus far has been on generating level design via machine learning [73, 195, 72, 79, 74, 76]. To this particular area I contribute a novel approach in the probabilistic graphical models discussed in Chapter 3. This approach is also the only one that learns to generate structure that can contain all game entities, while most other approaches simplify the allowable game structure (for example, removing decoration). Further, to the best of my knowledge, none of these models have been validated for content evaluation or generation with a human subject study. Thus the human subject study is itself a contribution as a model for PCGML evaluation. One general drawback of PCGML is that it requires data of a type of level to generate new levels of that type. While there has been some attempts to recombine data from multiple games to create new types of levels [70, 71], these approaches haven't demonstrated the ability to represent unseen human-authored level types as I demonstrate in Chapter 5.

There has been significantly less work at attempting to explicitly learn mechanics from games via machine learning for the purposes of generation. There exists some prior work at learning affordances, local relationships between game entities [124] and some prior work at learning hybrid automata (finite state machines) for the player character [101]. However, this prior work focuses on learning these representations for a single game (Super Mario Bros.), and they cannot be used to create playable games. To the best of my knowledge,

the engine learning approach discussed in Chapter 4 is the only existing work that looks to learn an explicit model of all game mechanics capable of simulating the original game. My demonstration of its application to other games, even of a limited type in platformer games designed for the NES, is another contribution.

There have been other approaches to game generation, most notably the ANGELINA system [91]. However, all prior work has employed rules-based or search-based approaches that require a large amount of human authoring. For example, the recent Gemini system [171] requires one-hundred and sixty-three hand-authored constraints to generate arcade-like games. In comparison the game generation systems in Chapter 7 are based upon machine learning. This has its own limitations and challenges, for example the issue of semantic readability of the output, but represents an entirely new approach to this problem and a new area of research.

9.1.2 Computational Creativity

Computational creativity approaches have historically focused on single-domain applications. For example, computational creativity approaches that can only produce music or visual art [1]. There have been approaches, such as conceptual blending [44] that have been applied to a variety of domains such as stories and games [51, 69, 71], but they are less common. Thus my introduction of conceptual expansion and application of it into multiple domains stands as a clear contribution as a general computational creativity approach.

While combinational creativity has been previously identified as a type of creativity [3], previously there has not been an effort to collect or survey the different approaches for combinational creativity. Thus my identification of this class of approaches is another contribution to the broader area of computational creativity. Further, where previously there have been a variety of highly-specialized combinational creativity approaches I have demonstrated that conceptual expansion subsumes several of these prior approaches across several domains, while handling arbitrary input and continuous variables by default.

There has been a number of applications of conceptual blending, but it has been unclear in some prior cases how to evaluate these systems outside of time-consuming human subject studies. Thus my demonstrate of evaluating such systems in regards to existing human creative work, as I do with the Super Mario Bros.: The Lost Levels levels in Chapter 5, is another contribution to this field.

Recently there has been some computational creativity work that draws on machine learning as the source of it's knowledge [196]. However, the vast majority of the field still relies on rules-based or search-based approaches, which makes it difficult to determine whether the source of the creativity is in the human authoring or the behavior of the system. By introducing instead applying these approaches to machine learned models and representations I have allowed for a new vector in the study of artificial creativity.

9.2 Future Work

I identify three, high-level areas of focus for future work: mixed-initiative applications, the conceptual expansion optimization problem, and automated game playing.

9.2.1 Mixed-Initiative Applications

In Chapter 7 I demonstrated the potential for mixed-initiative applications of the conceptual expansion-based game generation. However, I did not evaluate this mixed-initiative approach formally, or demonstrate how it could work with anyone other than myself working with the system. I have already been investigating the possibility of mixed-initiative, co-creative tools in the domain of video game levels [197, 198] using standard procedural content generation via machine learning (PCGML) tools. It follows then to continue this research into the possibility of taking these lessons and making them more accessible to non-experts through the creation of tools.

One benefit of mixed-initiative approaches is the benefit of the human's semantic knowledge. As can be seen from the generated games, many do not make semantic sense and

come across as fairly surreal. For example, games with decorative floating strings of numbers. This makes determining which games were created by humans versus AI fairly straightforward. A human could potentially help constrain or correct the output space to only those conceptual expansions that made sense to other humans.

9.2.2 Conceptual Expansion Optimization Problem

Conceptual expansion creates an output space exponential in size to the number of variables across its input set. In all of the conceptual expansion applications in this document I employed a fairly straightforward greedy optimization to search from the initial conceptual expansion defined by the mapping. There are two major issues with this current approach.

First, the construction of this mapping varied between the different domains, requiring domain-specific knowledge and strategies to construct in each case. In an ideal world there would either be a domain-agnostic mapping construction approach in order to make this method more general, or a means of searching the space of potential conceptual expansions without the definition of an initial starting point from the mapping. Notably I did not explicitly test the importance of the mapping, but note that without it, the results would likely be more akin to the random combination baseline in Chapter 6.

Second, this optimization is fairly limited. In both of the primary domains the final conceptual expansion was reached after only a small number of iterations, meaning a relatively small section of the output space was searched. One potential would be to draw on an alternative optimization approach that might allow one to find even stronger conceptual expansions. For example, an evolutionary search or genetic algorithm approach might allow one to search a more varied space of potential conceptual expansions. Alternatively an approach like MIMIC [199] might allow one to find global maxima in the output space of conceptual expansion for given input. While both of these approaches would be more time consuming, they also have the potential to further improve these results.

9.2.3 Automated Game Playing

In recent years the area of automated game playing has garnered much more research attention [110, 119, 120, 121]. The engine learning approach described in Chapter 4 and the application of conceptual expansion as a transfer mechanism could both serve to benefit automated game playing. In the former case, there are many approaches that allow for perfect game playing in cases where an AI agent is capable of forward simulation, for example Monte-Carlo Tree Search (MCTS). If one could perfectly or nearly perfectly learn an engine for a game this could be used as a forward model for the purposes of simulation. Further, either this model or a deep neural network applied to automated game playing (like in Deep Q learning [110]) could potentially be transferred between games through the application of conceptual expansion as transfer learning, as in Chapter 8.

REFERENCES

[1] S. Colton, G. A. Wiggins, *et al.*, "Computational creativity: The final frontier?" In *Ecai*, Montpelier, vol. 12, 2012, pp. 21–26.

[2] F. Pereira, P. Norvig, and A. Halevy, "The unreasonable effectiveness of data," *IEEE Intelligent Systems*, vol. 24, no. undefined, pp. 8–12, 2009.

[3] M. A. Boden, "Computer models of creativity," *AI Magazine*, vol. 30, no. 3, p. 23, 2009.

[4] M. Boden, *The creative mind: Myths and mechanisms*. Psychology Press, 2004.

[5] G. Fauconnier and M. Turner, *The way we think: Conceptual blending and the mind's hidden complexities*. Basic Books, 2002.

[6] S. Ontañón and E. Plaza, "Amalgams: A formal approach for combining multiple case solutions," in *Case-Based Reasoning. Research and Development*, Springer, 2010, pp. 257–271.

[7] W. Wilke and R. Bergmann, "Techniques and knowledge used for adaptation during case-based problem solving," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, Springer, 1998, pp. 497–506.

[8] J. Goguen, "Mathematical models of cognitive space and time," in *Reasoning and Cognition: Proc. of the Interdisciplinary Conference on Reasoning and Cognition*, 2006, pp. 125–128.

[9] K. Badie and M. T. Mahmoudi, "Compositional adaptation in case-based reasoning based on the semantic relations between the components in the cases," in *2017 IEEE International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, IEEE, 2017, pp. 449–454.

[10] J. Goguen and D. F. Harrell, "Style as a choice of blending principles," *Style and Meaning in Language, Art Music and Design*, pp. 49–56, 2004.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[12] P. S. Churchland and T. J. Sejnowski, *The computational brain*. MIT press, 2016.

[13] OpenAI, *Openai five*, `https://blog.openai.com/openai-five/`.

[14] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[15] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural content generation via machine learning (pcgml)," *arXiv preprint arXiv:1702.00539*, 2017.

[16] A. Craft, B. Jeffrey, and M. Leibling, *Creativity in education*. A&C Black, 2001.

[17] M. A. Boden, "Creativity and artificial intelligence," *Artificial Intelligence*, vol. 103, no. 1-2, pp. 347–356, 1998.

[18] K. Haase, "Discovery systems: From am to cyrano," 1987.

[19] D. B. Lenat, "Am: An artificial intelligence approach to discovery in mathematics as heuristic search," STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, Tech. Rep., 1976.

[20] ——, "Eurisko: A program that learns new heuristics and domain concepts: The nature of heuristics iii: Program design and results," *Artificial intelligence*, vol. 21, no. 1-2, pp. 61–98, 1983.

[21] D. B. Lenat and J. S. Brown, "Why am and eurisko appear to work," *Artificial intelligence*, vol. 23, no. 3, pp. 269–294, 1984.

[22] H. Gust, U. Krumnack, K.-U. Kühnberger, and A. Schwering, "Analogical reasoning: A core of cognition.," *KI*, vol. 22, no. 1, pp. 8–12, 2008.

[23] S. Bhatta, A. Goel, and S. Prabhakar, "Innovation in analogical design: A model-based approach," in *Artificial Intelligence in Design'94*, Springer, 1994, pp. 57–74.

[24] B. Falkenhainer, K. D. Forbus, and D. Gentner, "The structure-mapping engine: Algorithm and examples," *Artificial intelligence*, vol. 41, no. 1, pp. 1–63, 1989.

[25] A. K. Goel and S. R. Bhatta, "Use of design patterns in analogy-based design," *Advanced Engineering Informatics*, vol. 18, no. 2, pp. 85–94, 2004.

[26] B. J. Wiltgen, "Incremental design revision in biologically inspired design," PhD thesis, Georgia Institute of Technology, 2018.

[27] A. K. Goel, D. A. McAdams, and R. B. Stone, *Biologically inspired design*. Springer, 2015.

[28] A. K. Goel, A. G. de Silva Garza, N. Grué, J. W. Murdock, M. M. Recker, and T Govindaraj, "Towards design learning environments—i: Exploring how devices work," in *International conference on intelligent tutoring systems*, Springer, 1996, pp. 493–501.

[29] A. K. Goel, "Design, analogy, and creativity," *IEEE expert*, vol. 12, no. 3, pp. 62–70, 1997.

[30] B. Smyth and M. T. Keane, "Adaptation-guided retrieval: Questioning the similarity assumption in reasoning," *Artificial intelligence*, vol. 102, no. 2, pp. 249–293, 1998.

[31] F. Bou, M. Schorlemmer, J. Corneli, D Gómez-Ramírez, E. Maclean, A. Smaill, and A. Pease, "The role of blending in mathematical invention," in *Proceedings of the Sixth International Conference on Computational Creativity June*, 2015, p. 55.

[32] R. L. De Mantaras, D. McSherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M. L. Maher, M. T COX, K. Forbus, *et al.*, "Retrieval, reuse, revision and retention in case-based reasoning," *The Knowledge Engineering Review*, vol. 20, no. 3, pp. 215–240, 2005.

[33] J. Fox and S. Clarke, "Exploring approaches to dynamic adaptation," in *Proceedings of the 3rd International DiscCoTec Workshop on Middleware-Application Interaction*, ACM, 2009, pp. 19–24.

[34] R. Hervás and P. Gervás, "Case-based reasoning for knowledge-intensive template selection during text generation," in *European Conference on Case-Based Reasoning*, Springer, 2006, pp. 151–165.

[35] G. Sizov, P. Öztürk, and A. Aamodt, "Evidence-driven retrieval in textual cbr: Bridging the gap between retrieval and reuse," in *International Conference on Case-Based Reasoning*, Springer, 2015, pp. 351–365.

[36] J Murdock and A. Goel, "Meta-case-based reasoning: Using functional models to adapt case-based agents," *Case-based reasoning research and development*, pp. 407–421, 2001.

[37] S. Konieczny, J. Lang, and P. Marquis, "Da2 merging operators," *Artificial Intelligence*, vol. 157, no. 1-2, pp. 49–79, 2004.

[38] L. Steels and J. De Beule, "Unify and merge in fluid construction grammar," *EELC*, vol. 4211, pp. 197–223, 2006.

[39] J. Cojan and J. Lieber, "Conservative adaptation in metric spaces," in *European Conference on Case-Based Reasoning*, Springer, 2008, pp. 135–149.

[40] ——, "Belief merging-based case combination.," in *ICCBR*, Springer, 2009, pp. 105–119.

[41] S. Konieczny and R. P. Pérez, "Logic based merging," *Journal of Philosophical Logic*, vol. 40, no. 2, pp. 239–270, 2011.

[42] J. Cojan and J. Lieber, "Belief revision-based case-based reasoning," in *Proceedings of the ECAI-2012 Workshop SAMAI: Similarity and Analogy-based Methods in AI*, 2012, pp. 33–39.

[43] P. Thagard and T. C. Stewart, "The aha! experience: Creativity through emergent binding in neural networks," *Cognitive science*, vol. 35, no. 1, pp. 1–33, 2011.

[44] G. Fauconnier, "Conceptual blending and analogy," *The analogical mind: Perspectives from cognitive science*, pp. 255–286, 2001.

[45] P. J. Bentley and D. W. Corne, "An introduction to creative evolutionary systems," in *Creative evolutionary systems*, Elsevier, 2002, pp. 1–75.

[46] P. W. Poon and J. N. Carter, "Genetic algorithm crossover operators for ordering applications," *Computers & Operations Research*, vol. 22, no. 1, pp. 135–147, 1995.

[47] T. D. Gwiazda, *Crossover for single-objective numerical optimization problems*. Tomasz Gwiazda, 2006, vol. 1.

[48] G. Fauconnier and M. Turner, "Conceptual integration networks," *Cognitive science*, vol. 22, no. 2, pp. 133–187, 1998.

[49] D. P. O'Donoghue, Y. Abgaz, D. Hurley, and F. Ronzano, "Stimulating and simulating creativity with dr inventor," in *Proceedings of the 6th ICCC*, 2015.

[50] I. Alhashim, H. Li, K. Xu, J. Cao, R. Ma, and H. Zhang, "Topology-varying 3d shape creation via structural blending," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 158, 2014.

[51] B. Li, A. Zook, N. Davis, and M. O. Riedl, "Goal-driven conceptual blending: A computational approach for creativity," in *Proceedings of the 2012 International Conference on Computational Creativity, Dublin, Ireland*, 2012, pp. 3–16.

[52] S. Manzano, S. Ontanón, and E. Plaza, "Amalgam-based reuse for multiagent case-based reasoning," in *International Conference on Case-Based Reasoning*, Springer, 2011, pp. 122–136.

[53] T. R. Besold and E. Plaza, "Generalize and blend: Concept blending based on generalization, analogy, and amalgams.," in *ICCC*, 2015, pp. 150–157.

[54] S. Ontanón, J. Zhu, and E. Plaza, "Case-based story generation through story amalgamation," in *Proceedings of the ICCBR 2012 Workshops*, 2012, pp. 223–232.

[55] J. H. Holland, *Induction: Processes of inference, learning, and discovery*. Mit Press, 1989.

[56] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. Cheng, "A taxonomy of compositional adaptation," *Rapport Technique numéroMSU-CSE-04-17*, 2004.

[57] S. Eisenbach, C. Sadler, and D. Wong, "Component adaptation in contemporary execution environments," in *Distributed Applications and Interoperable Systems*, Springer, 2007, pp. 90–103.

[58] G. Müller and R. Bergmann, "Compositional adaptation of cooking recipes using workflow streams," in *Computer cooking contest, workshop proceedings ICCBR*, 2014.

[59] N. Reyhani, K. Badie, and M. Kharrat, "A new approach to compositional adaptation based on optimizing the global distance function and its application in an intelligent tutoring system," in *Information Reuse and Integration, 2003. IRI 2003. IEEE International Conference on*, IEEE, 2003, pp. 285–290.

[60] Z. Chedrawy and S. R. Abidi, "Case based reasoning for information personalization: Using a context-sensitive compositional case adaptation approach," in *Engineering of Intelligent Systems, 2006 IEEE International Conference on*, IEEE, 2006, pp. 1–6.

[61] M. Hendrikx, S. Meijer, J. V. D. Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Trans. Graph.*, vol. 9, no. 1, 1:1–1:22, Feb. 2013.

[62] J. Togelius, G. Yannakakis, K. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 3, no. 3, pp. 172–186, 2011.

[63] A. Liapis, G. N. Yannakakis, and J. Togelius, "Computational game creativity.," in *ICCC*, Citeseer, 2014, pp. 46–53.

[64] J. Dormans and S. Leijnen, "Combinatorial and exploratory creativity in procedural content generation," 2013.

[65] P. Ribeiro, F. C. Pereira, B. Marques, B. Leitao, and A. Cardoso, "A model for creativity in creature generation.," in *Proceedings of the 4th GAME-ON Conference*, 2003, p. 175.

[66] J Martins, F Pereira, E Miranda, and A. Cardoso, "Enhancing sound design with conceptual blending of sound descriptors," in *Proceedings of the 1st joint workshop on computational creativity*, 2004.

[67] M. Treanor, B. Blackford, M. Mateas, and I. Bogost, "Game-O-Matic: Generating videogames that represent ideas," in *Proceedings of the 3rd FDG working on Procedural Content Generation*, 2012.

[68] J. Gow and J. Corneli, "Towards generating novel games using conceptual blending," in *Proceedings of the 11th AIIDE Conference*, 2015.

[69] J. Permar and B. Magerko, "A conceptual blending approach to the generation of cognitive scripts for interactive narrative," in *Proceedings of the 9th AIIDE Conference*, 2013.

[70] S. Snodgrass and S. Ontañón, "An approach to domain transfer in procedural content generation of two-dimensional videogame levels," in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[71] A. Sarkar and S. Cooper, "Blending levels from different games using lstms," 2018.

[72] S. Snodgrass and S. Ontanon, "Learning to generate video game maps using markov models," *IEEE Transactions on Computational Intelligence and AI in Games*, 2016.

[73] A. J. Summerville, S. Philip, and M. Mateas, "Mcmcts pcg 4 smb: Monte carlo tree search to guide platformer level generation," in *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, AIIDE, 2015.

[74] A. Summerville and M. Mateas, "Super mario as a string: Platformer level generation via lstms," in *Proceedings of the First International Conference of DiGRA and FDG*, 2016.

[75] R. Jain, A. Isaksen, C. Holmgård, and J. Togelius, "Autoencoders for level generation, repair, and recognition," in *ICCC Workshop on Computational Creativity and Games*, vol. 306, 2016.

[76] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, "Evolving mario levels in the latent space of a deep convolutional generative adversarial network,"

in *Proceedings of the Genetic and Evolutionary Computation Conference*, ACM, 2018, pp. 221–228.

[77] S. Dahlskog and J. Togelius, "A multi-level level generator," in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, IEEE, Aug. 2014, pp. 1–8.

[78] A. K. Hoover, J. Togelius, and G. N. Yannakis, "Composing video game levels with music metaphors through functional scaffolding," in *First Computational Creativity and Games Workshop*, ACC, 2015.

[79] A. J. Summerville, S. Snodgrass, M. Mateas, and S. Ontanón, "The vglc: The video game level corpus," *arXiv preprint arXiv:1606.07487*, 2016.

[80] M. J. Nelson, J. Togelius, C. B. Browne, and M. Cook, "Rules and mechanics," in *Procedural Content Generation in Games (Computational Synthesis and Creative Systems)*, Springer, 2016, pp. 99–121.

[81] J. Togelius, "A procedural critique of deontological reasoning," in *Proceedings of DiGRA*, 2011.

[82] M. Cook, S. Colton, A. Raad, and J. Gow, "Mechanic miner: Reflection-driven game mechanic discovery and level design," in *European Conference on the Applications of Evolutionary Computation*, Springer, 2013, pp. 284–293.

[83] A. Zook and M. O. Riedl, "Automatic game design via mechanic generation.," in *AAAI*, 2014, pp. 530–537.

[84] B. Pell, "Metagame in symmetric chess-like games," 1992.

[85] V. Hom and J. Marks, "Automatic design of balanced board games," in *Third Artificial Intelligence and Interactive Digital Entertainment*, 2007, pp. 25–30.

[86] J. Togelius and J. Schmidhuber, "An experiment in automatic game design.," in *CIG*, 2008, pp. 111–118.

[87] C. Browne and F. Maire, "Evolutionary game design," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 1–16, 2010.

[88] A. M. Smith and M. Mateas, "Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games," in *Proceedings of the sixth IEEE Conference on Computational Intelligence and Games*, IEEE, 2010, pp. 273–280.

[89] A. Khalifa, M. C. Green, D. Perez-Liebana, and J. Togelius, "General video game rule generation," in *Computational Intelligence and Games*, 2017.

[90] M. Treanor, B. Schweizer, I. Bogost, and M. Mateas, "The micro-rhetorics of game-o-matic," in *Seventh International Conference on the Foundations of Digital Games*, 2012.

[91] M. Cook, S. Colton, and J. Gow, "The angelina videogame design system, part i," *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.

[92] M. J. Nelson and M. Mateas, "Recombinable game mechanics for automated design support.," in *Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2008.

[93] T. S. Nielsen, G. A. Barros, J. Togelius, and M. J. Nelson, "General video game evaluation using relative algorithm performance profiles," in *European Conference on the Applications of Evolutionary Computation*, Springer, 2015, pp. 369–380.

[94] T. Schaul, "A video game description language for model-based or interactive learning," in *Computational Intelligence in Games*, 2013, pp. 1–8.

[95] M. Nelson, S. Colton, E. Powley, S. Gaudl, P. Ivey, R. Saunders, B. Perez Ferrer, and M. Cook, "Mixed-initiative approaches to on-device mobile game design," in *CHI Workshop on Mixed-Initiative Creative Interfaces*, 2016.

[96] D. Ha and J. Schmidhuber, "World models," *arXiv preprint arXiv:1803.10122*, 2018.

[97] S. A. Eslami, D. J. Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor, *et al.*, "Neural scene representation and rendering," *Science*, vol. 360, no. 6394, pp. 1204–1210, 2018.

[98] M. Guzdial, B. Harrison, B. Li, and M. Riedl, "Crowdsourcing open interactive narrative.," in *Tenth International Conference on the Foundations of Digital Games*, 2015.

[99] J. C. Osborn, A. Summerville, and M. Mateas, "Automated game design learning," in *Computational Intelligence and Games*, 2017, pp. 240–247.

[100] J. Osborn, A. Summerville, and M. Mateas, "Automatic mapping of nes games with mappy," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, ACM, 2017, p. 78.

[101] A. Summerville, J. Osborn, and M. Mateas, "Charda: Causal hybrid automata recovery via dynamic analysis," *26th International Joint Conference on Artificial Intelligence*, 2017.

[102] E. Kalogerakis, S. Chaudhuri, D. Koller, and V. Koltun, "A probabilistic model for component-based shape synthesis," *ACM Transactions on Graphics*, vol. 31, no. 4, 55:1–55:11, Jul. 2014.

[103] N. Fish, M. Averkiou, O. van Kaick, O. Sorkine-Hornung, D. Cohen-Or, and N. J. Mitra, "Meta-representation of shape families," *ACM Transactions on Graphics*, vol. 33, no. 4, 34:1–34:11, Jul. 2014.

[104] A. Emilien, U. Vimont, M.-P. Cani, P. Poulin, and B. Benes, "World- brush: Interactive example-based synthesis of procedural virtual worlds," *ACM Transactions on Graphics*, vol. 34, no. 4, 106:1–106:11, Jul. 2015.

[105] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *CoRR*, vol. abs/1207.4708, 2012.

[106] A. Summerville, M. Guzdial, M. Mateas, and M. Riedl, "Learning player tailored content from observation: Platformer level generation from video traces using lstms," in *The 12th Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[107] S. Snodgrass and S. Ontanon, "Leveraging multi-layer level representations for puzzle-platformer level generation," in *Proceedings of the 4th Experimental AI in Games Workshop*, AAA, 2017.

[108] J. Osborn, B. Samuel, A. Summerville, and M. Mateas, "Towards general rpg playing," in *Proceedings of the 4th Experimental AI in Games Workshop*, AAA, 2017.

[109] R. A. Brooks, *Cambrian intelligence: The early history of the new AI*. Mit Press Cambridge, MA, 1999, vol. 44.

[110] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," Deepmind Technologies, Tech. Rep., 2013, arXiv:1312.5602.

[111] D. Hafner, "Deep reinforcement learning from raw pixels in doom," *arXiv preprint arXiv:1610.02164*, 2016.

[112] J. Oh, V. Chockalingam, S. Singh, and H. Lee, "Control of memory, active perception, and action in minecraft," *arXiv preprint arXiv:1605.09128*, 2016.

[113] P. Singh, T. Lin, E. T. Mueller, G. Lim, T. Perkins, and W. L. Zhu, "Open mind common sense: Knowledge acquisition from the general public," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, Springer, 2002, pp. 1223–1237.

[114] C. F. Gettys and S. D. Fisher, "Hypothesis plausibility and hypothesis generation," *Organizational behavior and human performance*, vol. 24, no. 1, pp. 93–110, 1979.

[115] M. Ranzato, A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra, "Video (language) modeling: A baseline for generative models of natural videos," *arXiv preprint arXiv:1412.6604*, 2014.

[116] V. Vukotić, S.-L. Pintea, C. Raymond, G. Gravier, and J. Van Gemert, "One-step time-dependent future video frame prediction with a convolutional encoder-decoder neural network," *arXiv preprint arXiv:1702.04125*, 2017.

[117] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, "Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization," *arXiv preprint arXiv:1610.02391*, 2016.

[118] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[119] A. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," *Experimental Robotics IX*, pp. 363–372, 2006.

[120] A. Uriarte and S. Ontanón, "Automatic learning of combat models for rts games," in *11th Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

[121] A. E. Braylan and R. Miikkulainen, "Object-model transfer in the general video game domain," in *The 12th Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[122] M. Ersen and S. Sariel, "Learning behaviors of and interactions among objects through spatio–temporal reasoning," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 1, pp. 75–87, 2015.

[123] C. Martens, A. Summerville, M. Mateas, J. Osborn, S. Harmon, N. Wardrip-Fruin, and A. Jhala, "Proceduralist readings, procedurally," in *12th Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[124] A. Summerville, M. Behrooz, M. Mateas, and A. Jhala, "What does that ?-block do? learning latent causal affordances from mario play traces," in *1st AAAI Workshop on what's next for AI in games*, 2017.

[125] A. Summerville, J. Osborn, C. Holmgård, and D. W. Zhang, "Mechanics automatically recognized via interactive observation: Jumping," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, ACM, 2017, p. 25.

[126] Y. Xian, B. Schiele, and Z. Akata, "Zero-shot learning-the good, the bad and the ugly," *arXiv preprint arXiv:1703.04394*, 2017.

[127] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[128] H. Daumé III, "Frustratingly easy domain adaptation," *arXiv preprint arXiv:0907.1815*, 2009.

[129] C. H. Lampert, H. Nickisch, and S. Harmeling, "Learning to detect unseen object classes by between-class attribute transfer," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, IEEE, 2009, pp. 951–958.

[130] Y.-X. Wang and M. Hebert, "Learning to learn: Model regression networks for easy small sample learning," in *European Conference on Computer Vision*, Springer, 2016, pp. 616–634.

[131] B. Kulis, K. Saenko, and T. Darrell, "What you saw is not what you get: Domain adaptation using asymmetric kernel transforms," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, IEEE, 2011, pp. 1785–1792.

[132] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[133] O. Levy and S. Markovitch, *Teaching machines to learn by metaphors*. Technion-Israel Institute of Technology, Faculty of Computer Science, 2012.

[134] M. Norouzi, T. Mikolov, S. Bengio, Y. Singer, J. Shlens, A. Frome, G. S. Corrado, and J. Dean, "Zero-shot learning by convex combination of semantic embeddings," *arXiv preprint arXiv:1312.5650*, 2013.

[135] T. Mensink, E. Gavves, and C. G. Snoek, "Costa: Co-occurrence statistics for zero-shot classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 2441–2448.

[136] L. J. Ba, K. Swersky, S. Fidler, and R. Salakhutdinov, "Predicting deep zero-shot convolutional neural networks using textual descriptions.," in *International Conference on Computer Vision*, 2015, pp. 4247–4255.

[137] M. Elhoseiny, Y. Zhu, H. Zhang, and A. Elgammal, "Zero shot learning from noisy text description at part precision," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[138] J. H. Wong and M. J. Gales, "Sequence student-teacher training of deep neural networks," *International Speech Communication Association*, 2016.

[139] J. Li, M. L. Seltzer, X. Wang, R. Zhao, and Y. Gong, "Large-scale domain adaptation via teacher-student learning," *arXiv preprint arXiv:1708.05466*, 2017.

[140] I. Kuzborskij and F. Orabona, "Stability and hypothesis transfer learning," in *International Conference on Machine Learning*, 2013, pp. 942–950.

[141] S. Azadi, M. Fisher, V. G. Kim, Z. Wang, E. Shechtman, and T. Darrell, "Multi-content gan for few-shot font style transfer," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7564–7573.

[142] L. Clouâtre and M. Demers, "Figr: Few-shot image generation with reptile," *arXiv preprint arXiv:1901.02199*, 2019.

[143] B. Cheong and H. Teo, "Can we train dogs and humans at the same time? gans and hidden distributions," Stanford, Tech. Rep., 2018.

[144] I. Kuzborskij, F. Orabona, and B. Caputo, "From n to n+ 1: Multiclass transfer incremental learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 3358–3365.

[145] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "Icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[146] W.-L. Chao, S. Changpinyo, B. Gong, and F. Sha, "An empirical study and analysis of generalized zero-shot learning for object recognition in the wild," in *European Conference on Computer Vision*, Springer, 2016, pp. 52–68.

[147] S. K. Divvala, A. Farhadi, and C. Guestrin, "Learning everything about anything: Webly-supervised visual concept learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3270–3277.

[148] Y. Yao, J. Zhang, F. Shen, X. Hua, J. Xu, and Z. Tang, "Exploiting web images for dataset construction: A domain robust approach," *IEEE Transactions on Multimedia*, vol. 19, no. 8, pp. 1771–1784, 2017.

[149] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: From architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[150] S. Snodgrass and S. Ontañón, "Experiments in map generation using markov chains," *Proceedings of the 9th International Conference on Foundations of Digital Games*, vol. 14, 2014.

[151] M. Guzdial, S. Shah, and M. Riedl, "Towards automated let's play commentary," *arXiv preprint arXiv:1809.09424*, 2018.

[152] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, "Real-time computer vision with opencv," *Commun. ACM*, vol. 55, no. 6, pp. 61–69, Jun. 2012.

[153] D. T. Pham, S. S. Dimov, and C. D. Nguyen, "Selection of k in k-means clustering," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 219, no. 1, pp. 103–119, 2005. eprint: `http://pic.sagepub.com/content/219/1/103.full.pdf+html`.

[154] J. Dormans, "Adventures in level design: Generating missions and spaces for action adventure games," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, ACM, 2010, p. 1.

[155] J. Togelius, S. Karakovskiy, and N. Shaker, *2012 mario ai championship*, http://www.marioai.org/, 2012.

[156] M. J. Foley, "Fuzzy merges: Examples and techniques," in *Proceedings of the Twenty-Fourth Annual SAS Users Group*, 1999.

[157] P. Weyhrauch, "Guiding interactive fiction," PhD thesis, Ph. D. Dissertation, Carnegie Mellon University, 1997.

[158] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling player experience in super mario bros," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, IEEE, 2009, pp. 132–139.

[159] A. Drachen, L. E. Nacke, G. Yannakakis, and A. L. Pedersen, "Correlation between heart rate, electrodermal activity and player experience in first-person shooter games," in *Proceedings of the Fifth ACM SIGGRAPH Symposium on Video Games*, ACM, 2010, pp. 49–54.

[160] A. Zook, B. Harrison, and M. O. Riedl, "Monte-carlo tree search for simulation-based strategy analysis," in *Proceedings of the 10th Conference on the Foundations of Digital Games*, 2015.

[161] A. Canossa and G. Smith, "Towards a procedural evaluation technique: Metrics for level design," *The 10th International Conference on the Foundations of Digital Games*, p. 8, 2015.

[162] C. Holmgård, M. C. Green, A. Liapis, and J. Togelius, "Automated playtesting with procedural personas through mcts with evolved heuristics," *arXiv preprint arXiv:1802.06881*, 2018.

[163] M. Guzdial, N. Sturtevant, and B. Li, "Deep static and dynamic level analysis: A study on infinite mario," in *3rd Experimental AI in Games Workshop*, 2016.

[164] N. Liao, M. Guzdial, and M. Riedl, "Deep convolutional player modeling on log and level data," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, ACM, 2017, p. 41.

[165] J. Togelius, S. Karakovskiy, and R. Baumgarten, "The 2009 mario ai competition," in *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8.

[166] M. Minsky, "K-lines: A theory of memory," *Cognitive science*, vol. 4, no. 2, pp. 117–133, 1980.

[167] N. R. Sturtevant and M. J. Ota, "Exhaustive and semi-exhaustive procedural content generation," in *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2018.

[168] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, ACM, 2010, p. 4.

[169] J. Orwant, "Eggg: Automated programming for game generation," *IBM Systems Journal*, vol. 39, no. 3.4, pp. 782–794, 2000.

[170] S. E. Gaudl, M. J. Nelson, S. Colton, R. Saunders, E. J. Powley, P. Ivey, B. P. Ferrer, and M. Cook, "Exploring novel game spaces with fluidic games," *arXiv preprint arXiv:1803.01403*, 2018.

[171] A. Summerville, C. Martens, B. Samuel, J. Osborn, N. Wardrip-Fruin, and M. Mateas, "Gemini: Bidirectional generation and analysis of games via asp," in *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2018.

[172] M. Cook, S. Colton, and J. Gow, "The angelina videogame design system—part ii," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 3, pp. 254–266, 2016.

[173] P. Gervás, "Stories from games: Content and focalization selection in narrative composition," in *Actas del Primer Simposio Espanol de Entretenimiento Digital*, 2013, p. 25.

[174] A. Eigenfeldt and P. Pasquier, "Negotiated content: Generative soundscape composition by autonomous musical agents in coming together: Freesound.," in *ICCC*, 2011, pp. 27–32.

[175] Y. Netzer, D. Gabay, Y. Goldberg, and M. Elhadad, "Gaiku: Generating haiku with word associations norms," in *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, Association for Computational Linguistics, 2009, pp. 32–39.

[176] J. K. Haas, "A history of the unity game engine," 2014.

[177] M. Guzdial, D. Long, C. Cassion, and A. Das, "Visual procedural content generation with an artificial abstract artist," in *Proceedings of ICCC Computational Creativity and Games Workshop*, 2017.

[178] M. Cook, "Make something that makes something: A report on the first procedural generation jam.," in *ICCC*, 2015, pp. 197–203.

[179] W. Brendel and M. Bethge, "Approximating cnns with bag-of-local-features models works surprisingly well on imagenet," *arXiv preprint arXiv:1904.00760*, 2019.

[180] P. Sweetser and P. Wyeth, "Gameflow: A model for evaluating player enjoyment in games," *Computers in Entertainment (CIE)*, vol. 3, no. 3, pp. 3–3, 2005.

[181] R. Bernhaupt, M. Eckschlager, and M. Tscheligi, "Methods for evaluating games: How to measure usability and user experience in games?" In *Proceedings of the international conference on Advances in computer entertainment technology*, ACM, 2007, pp. 309–310.

[182] M. Csikszentmihalyi, S. Abuhamdeh, and J. Nakamura, "Flow," in *Flow and the foundations of positive psychology*, Springer, 2014, pp. 227–238.

[183] M. Guzdial and M. O. Riedl, "Combinatorial meta search," in *Proceedings of the 1st Knowledge Extraction from Games Workshop*, 2018.

[184] N. Co., "Annual report 2018," *Nintendo Co., Ltd.*, Mar. 31, 2018.

[185] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Citeseer*, 2009.

[186] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR09*, 2009.

[187] T. Furlanello, Z. C. Lipton, A. Amazon, L. Itti, and A. Anandkumar, "Born again neural networks," in *NIPS Workshop on Meta Learning*, 2017.

[188] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[189] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-ucsd birds-200-2011 dataset," California Institute of Technology, Tech. Rep. CNS-TR-2011-001, 2011.

[190] W. Zhang, J. Sun, and X. Tang, "Cat head detection-how to effectively exploit shape and texture features," in *European Conference on Computer Vision*, Springer, 2008, pp. 802–816.

[191] A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei, "Novel dataset for fine-grained image categorization," in *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, 2011.

[192] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi, "Fine-grained visual classification of aircraft," *arXiv preprint arXiv:1306.5151*, 2013.

[193] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[194] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, 2016, pp. 2234–2242.

[195] A. K. Hoover, J. Togelius, and G. N. Yannakis, "Composing video game levels with music metaphors through functional scaffolding," in *Proceedings of the ICCC Workshop on Computational Creativity and Games*, 2015.

[196] P. Karimi, M. L. Maher, K. Grace, and N. Davis, "A computational model for visual conceptual blends," *IBM Journal of Research and Development*, 2018.

[197] M. Guzdial, N. Liao, J. Chen, S.-Y. Chen, S. Shah, V. Shah, J. Reno, G. Smith, and M. O. Riedl, "Friend, collaborator, student, manager: How design of an ai-driven game level editor affects creators," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19, Glasgow, Scotland Uk: ACM, 2019, 624:1–624:13, ISBN: 978-1-4503-5970-2.

[198] M. Guzdial and M. Riedl, "An interaction framework for studying co-creative ai," *arXiv preprint arXiv:1903.09709*, 2019.

[199] J. S. De Bonet, C. L. Isbell Jr, and P. A. Viola, "Mimic: Finding optima by estimating probability densities," in *Advances in neural information processing systems*, 1997, pp. 424–430.